# Efficient Algorithms for Kernel Aggregation Queries

Tsz Nam Chan, Leong Hou U, *Member, IEEE*, Reynold Cheng, *Member, IEEE*, Man Lung Yiu and Shivansh Mittal

**Abstract**—Kernel functions support a broad range of applications that require tasks like density estimation, classification, regression or outlier detection. For these tasks, a common online operation is to compute the weighted aggregation of kernel function values with respect to a set of points. However, scalable aggregation methods are still unknown for typical kernel functions (e.g., Gaussian kernel, polynomial kernel, sigmoid kernel and additive kernels) and weighting schemes. In this paper, we propose a novel and effective bounding technique, by leveraging index structures, to speed up the computation of kernel aggregation. In addition, we extend our technique to additive kernel functions, including $\chi^2$, intersection, JS and Hellinger kernels, which are widely used in different communities, e.g., computer vision, medical science, Geoscience etc. To handle the additive kernel functions, we further develop the novel and effective bound functions to efficiently evaluate the kernel aggregation. Experimental studies on many real datasets reveal that our proposed solution KARL achieves at least one order of magnitude speedup over the state-of-the-art for different types of kernel functions.

**Index Terms**—KARL, Kernel functions, lower and upper bounds

## 1 Introduction

Kernel functions are widely used in different machine learning models, including kernel density estimation (for statistical analysis), kernel regression (for prediction and forecasting) and kernel classification (for data mining). These types of models are actively used in many applications, which are summarized in Table 1. For density-estimation-based application, astronomical scientists [19] utilize kernel density estimation for quantifying the galaxy density. For regression-based application, environmental scientists [45], [55], [42] utilize support vector regression to forecast the wind speed which helps in predicting the generated energy by wind power. For classification-based application, network security systems [7], [6] utilize kernel SVM to detect suspicious packets. Due to its wide range of applications, many types of open-source libraries, e.g., LibSVM [12] and Scikit-learn [40], also support above machine learning models, which can combine with different kernel functions.

In the above applications, a common online operation is to compute the following function:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \cdot \mathcal{K}(\mathbf{q}, \mathbf{p_i}) \tag{1}$$

where $\mathbf{q}$ is a query point, $P$ is a dataset of points, $w_i$ is scalar, and $\mathcal{K}(\mathbf{q}, \mathbf{p_i})$ denotes the kernel function between $\mathbf{q}$ and $\mathbf{p_i}$. Table 2 summarizes all kernel functions which are widely used in existing work [12], [40], [36], [4]. Both $\gamma$, $\beta$ and $deg$ are constants and $dist(\mathbf{q}, \mathbf{p})$ denotes the Euclidean distance between $\mathbf{q}$ and $\mathbf{p}$. In addition, we also denote $q_\ell$ and $p_\ell$ as the $\ell^{\text{th}}$ dimensional values of vectors $\mathbf{q}$ and $\mathbf{p}$ respectively and $d$ is the dimensionality of the vector.

TABLE 1: Applications of kernel-based machine learning models

| Model | Application |
|---|---|
| Kernel density estimation/ classification (KDE/KDC) [21], [20] | Galaxy density quantification [19] |
| | Particle Searching [15] |
| Support vector regression (SVR) [47] | Wind speed forecasting [45], [55], [42] |
| | Ecological modeling [54] |
| | Time series prediction [43], [46] |
| | Image detection [26] |
| 1-class support vector machine (OCSVM) [37] | Suspicious packet detection [7], [6] |
| | Time series anomaly detection [25], [33] |
| 2-class support vector machine (2CSVM) [47] | Suspicious packet detection [7], [6] |
| | Cancer detection [14], [30] |
| | Image classification [36], [13], [16] |
| | Pedestrian detection [5], [3], [4] |

TABLE 2: Kernel functions

| Kernel name | Function $\mathcal{K}(\mathbf{q}, \mathbf{p})$ |
|---|---|
| Gaussian | $\exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p})^2)$ |
| Polynomial | $(\gamma \mathbf{q} \cdot \mathbf{p} + \beta)^{deg}$ |
| Sigmoid | $tanh(\gamma \mathbf{q} \cdot \mathbf{p} + \beta)$ |
| $\chi^2$ | $\sum_{\ell=1}^{d} \frac{2 q_\ell p_\ell}{q_\ell + p_\ell}$ |
| Intersection | $\sum_{\ell=1}^{d} \min(q_\ell, p_\ell)$ |
| JS | $\sum_{\ell=1}^{d} \frac{1}{2} q_\ell \log_2(\frac{q_\ell + p_\ell}{q_\ell}) + \frac{1}{2} p_\ell \log_2(\frac{q_\ell + p_\ell}{p_\ell})$ |
| Hellinger | $\sum_{\ell=1}^{d} \sqrt{q_\ell p_\ell}$ |

Two types of online prediction queries, namely $\epsilon$KAQ and $\tau$KAQ, are adopted in different machine learning models. Figure

- *T. N. Chan, R. Cheng and S. Mittal are with the Department of Computer Science, The University of Hong Kong, Hong Kong.*
  *E-mail: {tnchan, ckcheng}@cs.hku.hk and shivansh@connect.hku.hk*
- *L. H. U is with the State Key Laboratory of Internet of Things for Smart City and the Department of Computer and Information Science, University of Macau, Macau.*
  *E-mail: ryanlhu@umac.mo*
- *M. L. Yiu is with the Department of Computing, Hong Kong Polytechnic Univertiy, Hong Kong.*
  *E-mail: csmlyiu@comp.polyu.edu.hk*

1 shows the visualization of $\epsilon$KAQ and $\tau$KAQ queries in one attribute ($5^{th}$ dimension) of shuttle sensor dataset [2] with Gaussian kernel function.

***Approximate kernel aggregation query ($\epsilon$KAQ):*** In the regression/ density-estimation-based models, for example: KDE and SVR, the relative error, $\epsilon$, is used, such that for every query $\mathbf{q}$, the returned approximate value $F_{approx}$ is within $1 \pm \epsilon$ of $\mathcal{F}_P(\mathbf{q})$, i.e.,

$$(1-\epsilon)\mathcal{F}_P(\mathbf{q}) \leq F_{approx} \leq (1+\epsilon)\mathcal{F}_P(\mathbf{q}) \tag{2}$$

***Thresholded kernel aggregation query ($\tau$KAQ):*** In the classification-based models, for example: KDC, OCSVM and 2CSVM, the threshold, $\tau$, is adopted, such that for every query $\mathbf{q}$, the returned result is either 1 or -1 which indicates whether $\mathcal{F}_P(\mathbf{q}) \geq \tau$.

The above queries are expensive as it takes $O(nd)$ time to compute $\mathcal{F}_P(\mathbf{q})$ online, where $d$ is the dimensionality of data points and $n$ is the cardinality of the dataset $P$. In the machine learning community, many recent literatures [29], [24], [27] also complain about the inefficiency issues for computing kernel aggregation, which are quoted as follows:

- *"Despite their successes, what makes kernel methods difficult to use in many large scale problems is the fact that computing the decision function is typically expensive, especially at prediction time."* [29]
- *"However, computing the decision function for the new test samples is typically expensive which limits the applicability of kernel methods to real-world applications."* [24]
- *"..., it has the disadvantage of requiring relatively large computations in the testing phase."* [27]

To address the above inefficiency issues, existing solutions are divided into two camps (cf. Table 3). The machine learning community tends to improve the response time by using heuristics [27], [24], [29], [32] (e.g., sampling points in $P$), which may affect the quality of the model (e.g., classification/prediction accuracy). The other camp, which we are interested in, aims to enhance the efficiency while preserving the quality of the model. The pioneering solutions in this category are [21], [20], albeit they are only applicable to queries with type I weighting (see Table 3). Their idea [21], [20] is to build an index structure on the point set $P$ offline, and then exploit index nodes to derive lower/upper bounds and attempt pruning for online queries.

TABLE 3: Types of weighting in $\mathcal{F}_P(\mathbf{q})$

| Type of weighting | Used in model | Techniques |
|---|---|---|
| Type I: identical, positive $w_i$ (most specific) | KDE/KDC | Quality-preserving solutions [21], [20] |
| Type II: positive $w_i$ (subsuming Type I) | OCSVM | Heuristics [32] |
| Type III: no restriction on $w_i$ (subsuming Types I, II) | SVR, 2CSVM | Heuristics [27], [24], [29] |

In our preliminary work [10], we propose **K**ernel **A**ggregation **R**apid **L**ibrary (KARL)[1], which provides a fast solution to support Gaussian, polynomial and sigmoid kernels (cf. Table 2) with different types of weighting (cf. Table 3). We also compare with two widely-used libraries, namely LibSVM [12] and Scikit-learn [40]. Implementation-wise, LibSVM is based on the sequential scan method, and Scikit-learn is based on the algorithm in [21] for query type I. We compare them with our proposal (KARL) in Table 4. As a remark, since Scikit-learn supports query types II and III via the wrapper of LibSVM [40], we remove those two query types from the row of Scikit-learn in Table 4. The features of KARL are: (i) it supports all three types of weightings as well as both $\epsilon$KAQ and $\tau$KAQ queries, (ii) it supports index structures, (iii) it yields much lower response time than existing libraries.

TABLE 4: Comparisons of libraries

| Library | Supported queries | Supported weightings | Support indexing | Response time |
|---|---|---|---|---|
| LibSVM [12] | $\tau$KAQ | Types II, III | no | high |
| Scikit-learn [40] | $\epsilon$KAQ | Type I | yes | high |
| KARL (this paper) | $\epsilon$KAQ, $\tau$KAQ | Types I, II, III | yes | low |

However, existing libraries, including LibSVM [12], Scikit-learn [40] and KARL (our preliminary work) [10] only focus on Gaussian, polynomial and sigmoid kernels. Except for these three famous kernels, another class of kernel functions, called additive kernels, including $\chi^2$, intersection, JS and Hellinger kernels, has attracted more attention in many application domains recently, e.g., machine learning [39], [41], [56] and computer vision [53], [36], [49]. In this work, we extend KARL to support all additive kernels.

Compared to our preliminary work [10], there are three new contributions in this work. First, we develop the new lower and upper bound functions, which are based on the monotonicity property of additive kernel functions. We further show that these two bound functions can be computed in $O(d)$ time, if each $q_\ell$ is within the given range $[q_\ell^{(min)}, q_\ell^{(max)}]$. This range can be specified during the offline stage. Second, we extend the linear bound functions in our preliminary work [10] to support those $q_\ell$, which are not within this range $[q_\ell^{(min)}, q_\ell^{(max)}]$. Third, we further conduct new experiments for (1) additive kernels, (2) regression models, which are not supported in our preliminary work [10]. Our experimental results show that our method can achieve at least one order of magnitude speedup compared with the state-of-the-art methods.

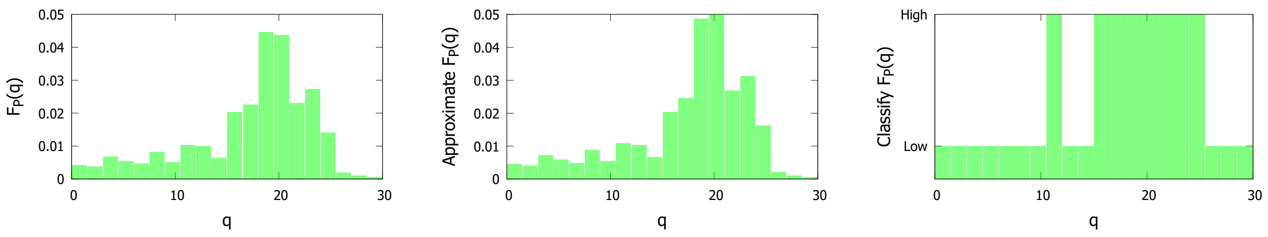
We first discuss the related work in Section 2. Later, we present the state-of-the-art method for evaluating $\epsilon$KAQ and $\tau$KAQ, using Gaussian kernel, in Section 3. Then, we discuss our solution KARL for Gaussian kernel in Section 4. Next, we extend our method KARL to handle additive kernel functions in Section 5. After that, we present our experiments in Section 6. Lastly, we conclude the paper with future research directions in Section 7.

## 2 Related Work

The term "kernel aggregation query" abstracts a common operation in several statistical and learning problems such as kernel density estimation [21], [20], 1-class SVM [37], 2-class SVM [47] and support vector regression [47].

Kernel density estimation is a non-parametric statistical method for density estimation. To speedup kernel density estimation, existing work would either compute approximate density values with accuracy guarantee [38] or test whether density values are above a given threshold [20]. Zheng et al. [58] focus on fast kernel density estimation on low-dimensional data (e.g., 1d, 2d)

1. https://github.com/edisonchan2013928/KARL-Fast-Kernel-Aggregation-Queries



(a) Exact KAQ (b) $\epsilon$KAQ with $\epsilon = 0.2$ (c) $\tau$KAQ with $\tau = 0.01$

Fig. 1: KAQ of the query region [0,30] in shuttle sensor dataset [2] (using the $5^{th}$ dimension)

and propose sampling-based solutions with theoretical guarantees on both efficiency and quality. On the other hand, [38], [20] assume that the point set $P$ is indexed by a $k$-d tree, and apply filter-and-refinement techniques for kernel density estimation. The library Scikit-learn [40] adopts the implementation in [38]. Our proposal, KARL, adapts the algorithm in [38], [20] to evaluate kernel aggregation queries. The key difference between KARL and [38], [20] lies in the bound functions. As explained in our preliminary work [10], our proposed linear bound functions are tighter than existing bound functions used in [38], [20]. Furthermore, we extend our linear bound functions to deal with different types of weighting and kernel functions [10], which have not been considered in [38], [20].

SVM is proposed by the machine learning community to classify data objects or detect outliers. SVM has been applied in different application domains, such as document classification [37], network fault detection [57], [7], [6], anomaly/outlier detection [11], [31], novelty detection [25], [33], [48], cancer detection [14], [30], image classification [13], [53], [36], time series classification [28], and pedestrian detection [4]. The typical process is divided into two phases. In the offline phase, training/learning algorithms are used to obtain the point set $P$, the weighting, and parameters. Then, in the online phase, thresholded kernel aggregation query ($\tau$KAQ) can be used to support classification or outlier detection. Two approaches have been studied to accelerate the online phase. The library LibSVM [12] assumes sparse data format and applies inverted index to speedup exact computation. The machine learning community has proposed heuristics [27], [24], [29], [32] to reduce the size of the point set $P$ in the offline phase, in order to speedup the online phase. However, these heuristics may affect the prediction quality of SVM. Our proposed bound functions have not been studied in the above work.

Most of the existing work, e.g., [47], [10], only focus on three kernel functions, including Gaussian, polynomial and sigmoid kernel functions. Recently, additive kernels (e.g., $\chi^2$, intersection, JS, Hellinger kernels in Table 2) with SVM are extensively used for different applications, e.g., image classification [53], [36] in computer vision, detection of spatial properties of object in Geoscience [16], colorectal cancer detection in medical science [30], pedestrian detection system in transportation [4]. However, it is still time-consuming (still in $O(nd)$ time) to evaluate $\epsilon$KAQ and $\tau$KAQ, using additive kernels. To boost the efficiency performance, many approximation methods have been developed in existing work, which can be divided into two camps.

In the first camp, researchers [34], [50], [49], [53] aim to learn the finite $D$-dimensional feature representation of each vector, e.g., $\phi(\mathbf{q})$ for $\mathbf{q}$ and $\phi(\mathbf{p})$ for $\mathbf{p}$, such that $\mathcal{K}(\mathbf{q},\mathbf{p}) \approx \phi(\mathbf{q})^T\phi(\mathbf{p})$ in which it can reduce the time complexity for evaluating kernel aggregation function from $O(nd)$ to $O(D)$ time in the online phase. However, this type of work only provides the approximation results for both $\epsilon$KAQ and $\tau$KAQ without any theoretical guarantee. In the second camp, researchers [23], [35], [36] discover that the kernel aggregation function can be evaluated exactly and efficiently with some additive kernels. For example: Maji et al. [35], [36] show that the kernel aggregation function can be computed in $O(d \log n)$ time using intersection kernel. However, not all additive kernels (e.g., $\chi^2$, JS and Hellinger) exhibit this property. To speedup the evaluation of kernel aggregation function for other kernels, some work [36], [5], [4] further prestore the kernel aggregation values into the lookup table, which can be used to evaluate the kernel aggregation function effectively and approximately. Even though this type of methods is similar with our proposal, it does not provide any theoretical guarantee. In our work, we further support these kernel functions for evaluating $\epsilon$KAQ and $\tau$KAQ, which guarantee our returned result within the relative error $\epsilon$ and correctly classify the value with the threshold $\tau$, respectively.

## 3 State-of-the-Art (SOTA) For Gaussian Kernel

In this section, we introduce the state-of-the-art [21], [20] (SOTA), albeit it is only applicable to queries with type I weighting (see Table 3) and Gaussian kernel function. In this case, we denote the common weight by $w$ and $\mathcal{K}(\mathbf{q},\mathbf{p_i}) = \exp(-\gamma \cdot dist(\mathbf{q},\mathbf{p_i})^2)$ in Equation 1.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i}\in P} w \cdot \exp(-\gamma \cdot dist(\mathbf{q},\mathbf{p_i})^2) \tag{3}$$

***Bounding functions.***

We introduce the concept of *bounding rectangle* [44] below.

**Definition 1.** *Let $R$ be the bounding rectangle for a point set $P$. We denote its interval in the $j$-th dimension as $[R[j].l, R[j].u]$, where $R[j].l = \min_{\mathbf{p}\in P} \mathbf{p}[j]$ and $R[j].u = \max_{\mathbf{p}\in P} \mathbf{p}[j]$.*

Given a query point $\mathbf{q}$, we can compute the minimum distance $mindist(\mathbf{q}, R)$ from $\mathbf{q}$ to $R$, and the maximum distance $maxdist(\mathbf{q}, R)$ from $\mathbf{q}$ to $R$, i.e., the following inequality holds for every point $\mathbf{p}$ inside $R$.

$$mindist(\mathbf{q}, R) \le dist(\mathbf{q},\mathbf{p}) \le maxdist(\mathbf{q}, R)$$

With the above notations, the lower bound $LB_R(\mathbf{q})$ and the upper bound $UB_R(\mathbf{q})$ for $\mathcal{F}_P(\mathbf{q})$ (Equation 3) are defined as:

$$\begin{aligned} LB_R(\mathbf{q}) &= w \cdot R.\texttt{count} \cdot \exp(-\gamma \cdot maxdist(\mathbf{q}, R)^2) \\ UB_R(\mathbf{q}) &= w \cdot R.\texttt{count} \cdot \exp(-\gamma \cdot mindist(\mathbf{q}, R)^2) \end{aligned}$$

where $R.\texttt{count}$ denotes the number of points (from $P$) in $R$, and $w$ denotes the common weight (for type I weighting). It takes $O(d)$ time to compute the above bounds online.

***Refinement of bounds.***

The state-of-the-art [21], [20] employs a hierarchical index structure (e.g., $k$-d tree) to index the point set $P$. Consider the example index in Figure 2. Each non-leaf entry (e.g., $R_5, 9$) stores the bounding rectangle of its subtree (e.g., $R_5$) and the number of points in its subtree (e.g., 9).

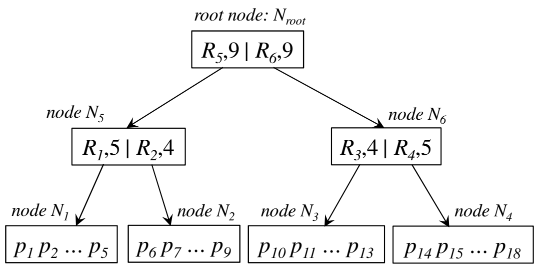


Fig. 2: Hierarchical index structure

We illustrate the running steps of the state-of-the-art on the above example index in Table 5. For conciseness, the notations $LB_R(\mathbf{q}), UB_R(\mathbf{q}), \mathcal{F}_P(\mathbf{q})$ are abbreviated as $lb_R, ub_R, \mathcal{F}_P$ respectively. The state-of-the-art maintains a lower bound $\widehat{lb}$ and upper bound $\widehat{ub}$ for $\mathcal{F}_P(\mathbf{q})$. Initially, the bounding rectangle of the root node (say, $R_{root}$) is used to compute $\widehat{lb}$ and $\widehat{ub}$. It uses a priority queue to manage the index entries that contribute to the computation of those bounds; the priority of an index entry $R_i$ is defined as the difference $ub_{R_i} - lb_{R_i}$. In each iteration, the algorithm pops an entry $R_i$ from the priority queue, processes the child entries of $R_i$, then refines the bounds incrementally and updates the priority queue. For example, in step 2, the algorithm pops the entry $R_5$ from the priority queue, inserts its child entries $R_1, R_2$ into the priority queue, and refines the bounds incrementally. Similar technique can be also found in similarity search community (e.g., [8], [9]).

TABLE 5: Running steps for state-of-the-art

| Step | Priority queue | Maintenance of lower bound $\widehat{lb}$ and upper bound $\widehat{ub}$ |
|---|---|---|
| 1 | $R_{root}$ | $\widehat{lb} = lb_{R_{root}}$, $\widehat{ub} = ub_{R_{root}}$ |
| 2 | $R_5, R_6$ | $\widehat{lb} = lb_{R_5} + lb_{R_6}$, $\widehat{ub} = ub_{R_5} + ub_{R_6}$ |
| 3 | $R_6, R_1, R_2$ | $\widehat{lb} = lb_{R_6} + lb_{R_1} + lb_{R_2}$, $\widehat{ub} = ub_{R_6} + ub_{R_1} + ub_{R_2}$ |
| 4 | $R_1, R_2, R_3, R_4$ | $\widehat{lb} = lb_{R_1} + lb_{R_2} + lb_{R_3} + lb_{R_4}$, $\widehat{ub} = ub_{R_1} + ub_{R_2} + ub_{R_3} + ub_{R_4}$ |
| 5 | $R_2, R_3, R_4$ | $\widehat{lb} = \mathcal{F}_{p_1 \cdots p_5} + lb_{R_2} + lb_{R_3} + lb_{R_4}$, $\widehat{ub} = \mathcal{F}_{p_1 \cdots p_5} + ub_{R_2} + ub_{R_3} + ub_{R_4}$ |

The state-of-the-art terminates upon reaching a termination condition. For $\tau$KAQ, the termination condition is: $\widehat{lb} \geq \tau$ or $\widehat{ub} < \tau$. For $\epsilon$KAQ, the termination condition is: $\widehat{ub} < (1+\epsilon)\widehat{lb}$.

## 4 KARL FOR GAUSSIAN KERNEL

Our proposed solution, KARL, adopts the state-of-the-art (SOTA) for query processing, except that existing bound functions (e.g., $LB_R(\mathbf{q})$ and $UB_R(\mathbf{q})$) are replaced by our bound functions.

Our key contribution is to develop tighter bound functions for $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 3). In Section 4.1, we propose a novel idea to bound the function $\exp(-x)$ and discuss how to compute such bound functions quickly. In Section 4.2, we devise tighter bound functions and show that they are always tighter than existing bound functions.

In this section, we only focus on the Gaussian kernel in the function $\mathcal{F}_P(\mathbf{q})$. As a remark, our preliminary work [10] proposes some advanced techniques, e.g., auto-tuning, to further boost the efficiency performance. In addition, we also propose a method to support both polynomial and sigmoid kernels in [10]. To save space, we omit these parts. Interested readers can refer to [10].

### 4.1 Fast Linear Bound Functions

We wish to design bound functions such that (i) they are tighter than existing bound functions (cf. Section 3), and (ii) they are efficient to compute, i.e., taking only $O(d)$ computation time.

In this section, we assume type I weighting and denote the common weight by $w$. Consider an example on the dataset $P = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}\}$. Let $x_i$ be the value $\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2$. With this notation, the value $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 3) can be simplified to:

$$w\Big(\exp(-x_1) + \exp(-x_2) + \exp(-x_3)\Big).$$

In Figure 3, we plot the function value $\exp(-x)$ for $x_1, x_2, x_3$ as points.

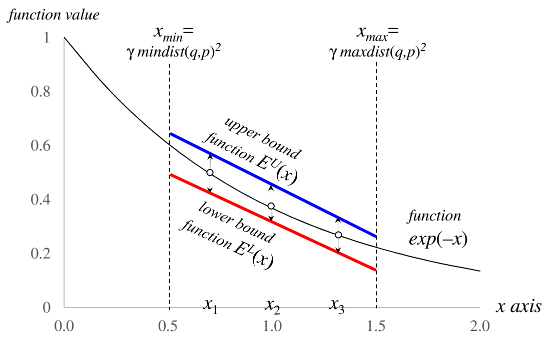


Fig. 3: Linear bounds

We first sketch our idea for bounding $\mathcal{F}_P(\mathbf{q})$. First, we compute the bounding interval of $x_i$, i.e., the interval $[x_{min}, x_{max}]$, where $x_{min} = \gamma \cdot mindist(\mathbf{q}, R)^2$, $x_{max} = \gamma \cdot maxdist(\mathbf{q}, R)^2$, and $R$ is the bounding rectangle of $P$. Within that interval, we employ two functions $E^L(x)$ and $E^U(x)$ as lower and upper bound functions for $\exp(-x)$, respectively (cf. Definition 2). We illustrate these two functions by a red line and a blue line in Figure 3.

**Definition 2** (Constrained bound functions)**.** *Given a query point* $\mathbf{q}$ *and a point set* $P$*, we call two functions* $E^L(x)$ *and* $E^U(x)$ *to be lower and upper bound functions for* $\exp(-x)$*, respectively, if*

$$E^L(x) \leq \exp(-x) \leq E^U(x)$$

*holds for any* $x \in [\gamma \cdot mindist(\mathbf{q}, R)^2, \gamma \cdot maxdist(\mathbf{q}, R)^2]$*, where* $R$ *is the bounding rectangle of* $P$*.*

In this paper, we model bound functions $E^L(x)$ and $E^U(x)$ by using two *linear functions* $Lin_{m_l,c_l}(x) = m_l x + c_l$ and

$Lin_{m_u,c_u}(x) = m_u x + c_u$, respectively. Then, we define the aggregation of a linear function $Lin_{m,c}(x)$ as:

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = \sum_{\mathbf{p_i} \in P} w\Big(m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c\Big) \quad (4)$$

With this concept, the functions $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ and $\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u})$ serve as a lower and an upper bound function for $\mathcal{F}_P(\mathbf{q})$ respectively, subject to the condition stated in Lemma 1.

**Lemma 1.** *Suppose that $Lin_{m_l,c_l}(x)$ and $Lin_{m_u,c_u}(x)$ are lower and upper bound functions for $\exp(-x)$, respectively, for the query point $\mathbf{q}$ and point set $P$. It holds that:*

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l}) \le \mathcal{F}_P(\mathbf{q}) \le \mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u}) \quad (5)$$

Observe that the bound functions in Figure 3 are not tight. We will devise tighter bound functions in the next subsection.

***Fast computation of bounds.***

The following lemma allows $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ to be efficiently computed in $O(d)$ time.

**Lemma 2.** *Given two values $m$ and $c$, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ (Equation 4) can be computed in $O(d)$ time and it holds that:*

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = wm\gamma\Big(|P| \cdot ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{a_P} + b_P\Big) + wc|P|$$

*where $\mathbf{a_P} = \sum_{\mathbf{p_i} \in P} \mathbf{p_i}$ and $b_P = \sum_{\mathbf{p_i} \in P} ||\mathbf{p_i}||^2$.*

*Proof.*

$$\begin{aligned}\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) &= \sum_{\mathbf{p_i} \in P} w\Big(m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c\Big) \\ &= wm\gamma \sum_{\mathbf{p_i} \in P} \Big(||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{p_i} + ||\mathbf{p_i}||^2\Big) + wc|P| \\ &= wm\gamma\Big(|P| \cdot ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{a_P} + b_P\Big) + wc|P|\end{aligned}$$

Observe that both terms $\mathbf{a_P}$ and $b_P$ are independent of the query point $\mathbf{q}$. Therefore, with the pre-computed values of $\mathbf{a_P}$ and $b_P$, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ can be computed in $O(d)$ time. □

## 4.2 Tighter Bound Functions

We proceed to devise tighter bound functions by using $Lin_{m_l,c_l}(x)$ and $Lin_{m_u,c_u}(x)$.

***Linear function $Lin_{m_u,c_u}(x)$ for modeling $E^U(x)$.***

Observe from Figure 4, the chord-based linear bound function $Lin_{m_u,c_u}(x)$, which passes through two points $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$, can act as the proper upper bound function of $\exp(-x)$, given each $x_i$ is in the interval $[x_{min}, x_{max}]$.

It turns out that $Lin_{m_u,c_u}(x)$ can also lead to a tighter upper bound than the existing bound $\exp(-x_{min})$ (see Section 3), as shown in Figure 4 (green dashed line in Figure 4), which is stated in Lemma 3.

**Lemma 3.** *There exists a linear function $Lin_{m_u,c_u}(x) = m_u x + c_u$ with:*

$$\begin{aligned} m_u &= \frac{\exp(-x_{max}) - \exp(-x_{min})}{x_{max} - x_{min}} & (6) \\ c_u &= \frac{x_{max}\exp(-x_{min}) - x_{min}\exp(-x_{max})}{x_{max} - x_{min}} & (7)\end{aligned}$$

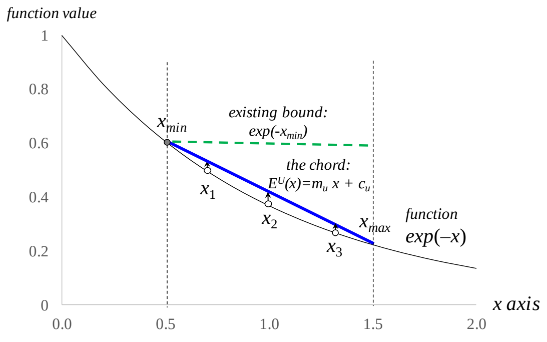


Fig. 4: Chord-based upper bound function

*such that $\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u}) \le UB_R(\mathbf{q})$, where $UB_R(\mathbf{q})$ is the upper bound function used in the state-of-the-art (see Section 3).*

***Linear function $Lin_{m_l,c_l}(x)$ for modeling $E^L(x)$.***

To achieve the (1) correct and (2) tight lower bound for $\exp(-x)$, we utilize the tangent line, which passes through the tangent point, as the correct lower bound of $\exp(-x)$ (cf. Figure 5), due to the convexity property of this function. Observe from Figure 5a, once we choose the tangent point $(x_{max}, \exp(-x_{max}))$, the lower bound is already tighter than the existing bound $\exp(-x_{max})$ (green dashed line in Figure 5a), as stated in Lemma 4.

**Lemma 4.** *There exists a linear function $Lin_{m_l,c_l}(x)$ such that $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l}) \ge LB_R(\mathbf{q})$, where $LB_R(\mathbf{q})$ is the lower bound function used in the state-of-the-art (see Section 3).*

Interestingly, it is possible to devise a tighter bound than the above. Figure 5b depicts the tangent line at point $(t, \exp(-t))$. This tangent line offers a much tighter bound than the one in Figure 5a.

In the following, we demonstrate how to find the optimal tangent line (i.e., leading to the tightest bound). Suppose that the linear function of lower bound $Lin_{m_l,c_l}(x)$ is the tangent line at point $(t, \exp(-t))$. Then, we derive the slope $m_l$ and the intercept $c_l$ as:

$$\begin{aligned} m_l &= \frac{d\exp(-x)}{dx}\Big|_{x=t} = -\exp(-t) \\ c_l &= \exp(-t) - m_l t = (1+t)\exp(-t)\end{aligned}$$

Theorem 1 establishes the optimal value $t_{opt}$ that leads to the tightest bound.

**Theorem 1.** *Consider the function $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ as a function of $t$, where $m_l = -\exp(-t)$ and $c_l = (1+t)\exp(-t)$. This function yields the maximum value at:*

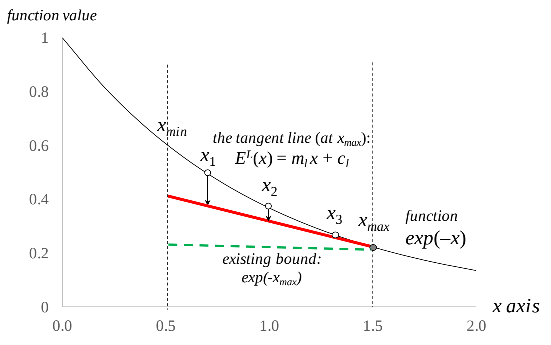
$$t_{opt} = \frac{\gamma}{|P|} \cdot \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \quad (8)$$

*Proof.* Let $H(t) = \mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ be a function of $t$. For the sake of clarity, we define the following two constants that are independent of $t$:

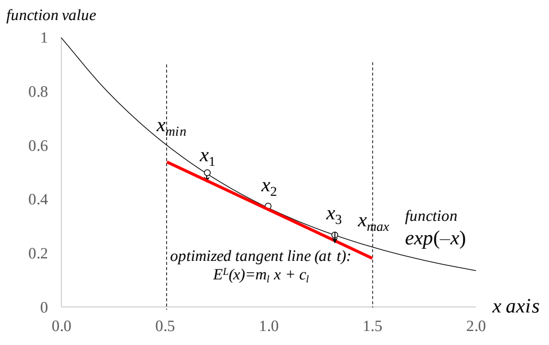
$$z_1 = w\gamma \cdot \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \text{ and } z_2 = w|P|$$

Together with the given $m_l$ and $c_l$, we can rewrite $H(t)$ as:

$$H(t) = -z_1 \exp(-t) + z_2(1+t)\exp(-t)$$



(a) tangent line at $x_{max}$



(b) optimized tangent line at $t$

Fig. 5: Tangent-based lower bound function

The remaining proof is to find the maximum value of $H(t)$. We first compute the first derivative of $H(t)$ (in terms of $t$):

$$\begin{aligned} H'(t) &= z_1 \exp(-t) + z_2 \exp(-t) - z_2(1+t)\exp(-t) \\ &= (z_1 + z_2 - z_2 - z_2 t)\exp(-t) \\ &= (z_1 - z_2 t)\exp(-t) \end{aligned}$$

Next, we compute the value $t_{opt}$ such that $H'(t_{opt}) = 0$. Since $\exp(-t_{opt}) \neq 0$, we get:

$$\begin{aligned} z_1 - z_2 t_{opt} &= 0 \\ t_{opt} &= \frac{z_1}{z_2} = \frac{\gamma}{|P|} \cdot \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \end{aligned}$$

Then we further test whether $t_{opt}$ indeed yields the maximum value. We consider two cases for $H'(t)$. Note that both $z_1$ and $z_2$ are positive constants.

1) For the case $t < t_{opt}$, we get $H'(t) > 0$, implying that $H(t)$ is an increasing function.
2) For the case $t > t_{opt}$, we get $H'(t) < 0$, implying that $H(t)$ is a decreasing function.

Thus, we conclude that the function $H(t)$ yields the maximum value at $t = t_{opt}$. □

The optimal value $t_{opt}$ involves the term $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$. This term can be computed efficiently in $O(d)$ time by the trick in Lemma 2. By applying Lemma 2 and substituting $w = m = \gamma = 1$ and $c = 0$, we can express $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$ in the form of $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$, which can be computed in $O(d)$ time.

***Case study.***

We conduct the following case study on the augmented $k$-d tree, in order to demonstrate the performance of KARL and the tightness of our bound functions compared to existing bound functions. First, we pick a random query point from the home dataset [2] (see Section 6.1 for details). Then, we plot the lower/upper bound values of SOTA and KARL versus the number of iterations. Observe that our bounds are much tighter than existing bounds, and thus KARL terminates sooner than SOTA.

## 4.3 Other Types of Weighting

The state-of-the-art solution [21], [20] only considers type-I weighting. However, as stated in Table 3, different types of weighting are adopted in different statistical/ machine learning models. In this section, we extend our bounding techniques for the following function under other types of weighting:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \cdot \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2)$$

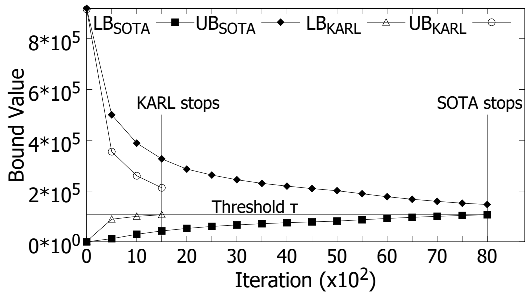


Fig. 6: Bound values of SOTA and KARL vs. the number of iterations; for type I-$\tau$ query on the home dataset

### 4.3.1 *Type II Weighting*

For type II weighting, each $w_i$ takes a positive value. Note that different $w_i$ may take different values.

First, we redefine the aggregation of a linear function $Lin_{m,c}(x)$ as:

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = \sum_{\mathbf{p_i} \in P} w_i \Big( m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c \Big) \quad (9)$$

This function can also be computed efficiently (i.e., in $O(d)$ time), as shown in Lemma 5.

**Lemma 5.** *Under type II weighting, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ (Equation 9) can be computed in $O(d)$ time, given two values of $m$ and $c$.*

*Proof.*

$$\begin{aligned} & \mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) \\ = & \sum_{\mathbf{p_i} \in P} w_i \Big( m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c \Big) \\ = & \sum_{\mathbf{p_i} \in P} w_i \Big( m\gamma \Big( ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{p_i} + ||\mathbf{p_i}||^2 \Big) \Big) + c \sum_{\mathbf{p_i} \in P} w_i \\ = & m\gamma \Big( w_P \cdot ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{a_P} + b_P \Big) + c w_P \end{aligned}$$

where $\mathbf{a_P} = \sum_{\mathbf{p_i} \in P} w_i \mathbf{p_i}$, $b_P = \sum_{\mathbf{p_i} \in P} w_i ||\mathbf{p_i}||^2$ and $w_P = \sum_{\mathbf{p_i} \in P} w_i$.

The terms $\mathbf{a_P}, b_P, w_P$ are independent of $\mathbf{q}$. With their pre-computed values, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ can be computed in $O(d)$ time. □

It remains to discuss how to find tight bound functions. For the upper bound function, we adopt the same technique in Figure 4. For the lower bound function, we use the idea in Figure 5b, except that the optimal value $t_{opt}$ should also depend on the weighting (cf. Theorem 2).

**Theorem 2.** *Consider the function $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ as a function of t, where $m_l = -\exp(-t)$ and $c_l = (1+t)\exp(-t)$. This function yields the maximum value at:*

$$t_{opt} = \frac{\gamma}{w_P} \cdot \sum_{\mathbf{p_i} \in P} w_i \cdot dist(\mathbf{q}, \mathbf{p_i})^2 \tag{10}$$

*where $w_P = \sum_{\mathbf{p_i} \in P} w_i$.*

*Proof.* Following the proof of Theorem 1, we let $H(t) = \mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ be a function of $t$ and we define the following two constants.

$$z_1 = \gamma \cdot \sum_{\mathbf{p_i} \in P} w_i \cdot dist(\mathbf{q}, \mathbf{p_i})^2 \text{ and } z_2 = w_P$$

Then, we follow exactly the same steps of Theorem 1 to derive the optimal $t_{opt}$ (Equation 10). □

Again, the value $t_{opt}$ can also be computed efficiently (i.e., in $O(d)$ time).

### 4.3.2 Type III Weighting

For type III weighting, there is no restriction on $w_i$. Each $w_i$ takes either a positive value or a negative value.

Our idea is to convert the problem into two subproblems that use type II weighting. First, we partition the point set $P$ into two sets $P^+$ and $P^-$ such that: (i) all points in $P^+$ are associated with positive weights, and (ii) all points in $P^-$ are associated with negative weights. Then we introduce the following notation:

$$\underline{\mathcal{F}_{P^-}}(\mathbf{q}) = \sum_{\mathbf{p_i} \in P^-} |w_i| \cdot \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) = -\mathcal{F}_{P^-}(\mathbf{q})$$

This enables us to rewrite the function $\mathcal{F}_P(\mathbf{q})$ as:

$$\begin{aligned} \mathcal{F}_P(\mathbf{q}) &= \sum_{\mathbf{p_i} \in P^+ \cup P^-} w_i \cdot \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) \\ &= \mathcal{F}_{P^+}(\mathbf{q}) + \mathcal{F}_{P^-}(\mathbf{q}) \\ &= \mathcal{F}_{P^+}(\mathbf{q}) - \underline{\mathcal{F}_{P^-}}(\mathbf{q}) \end{aligned}$$

Since the weights in both $\mathcal{F}_{P^+}(\mathbf{q})$ and $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$ are positive, the terms $\mathcal{F}_{P^+}(\mathbf{q})$ and $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$ can be bounded by using the techniques for type II weighting.

The upper bound of $\mathcal{F}_P(\mathbf{q})$ can be computed as the upper bound of $\mathcal{F}_{P^+}(\mathbf{q})$ minus the lower bound of $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$.

The lower bound of $\mathcal{F}_P(\mathbf{q})$ can be computed as the lower bound of $\mathcal{F}_{P^+}(\mathbf{q})$ minus the upper bound of $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$.

## 5 KARL FOR ADDITIVE KERNELS

In previous sections, we mainly focus on Gaussian kernel function. However, as discussed in Introduction (cf. Section 1), additive kernel functions (e.g., $\chi^2$, intersection, JS and Hellinger kernels in Table 2) have recently attracted the attention in both machine learning [56], [17], [41], [39] and computer vision [35], [49], [36], [53] communities, which are actively used in the following applications. Some pedestrian detection systems [5], [3], [4], utilize additive kernels to detect human. Medical scientists [30] utilize additive kernels to detect colorectal cancer. Geoscientists [16] utilize additive kernels to characterize spatial properties of objects in a scene. However, the same as Gaussian kernel, evaluating kernel aggregation function $\mathcal{F}_P(\mathbf{q})$ with additive kernels also takes $O(nd)$ time, which is slow in the prediction phase of machine learning models. Therefore, we extend KARL to support this class of kernel functions.

Additive kernel functions exhibit the following additive property (cf. Definition 3).

**Definition 3.** *(Additive property [36]) We denote $\mathcal{K}(\mathbf{q}, \mathbf{p})$ as the additive kernel, if we can express this function as the sum of d one-dimensional kernel functions (denoted as $K_\ell(q_\ell, p_\ell)$, where $1 \le \ell \le d$), which correspond to different dimensions, where:*

$$\mathcal{K}(\mathbf{q}, \mathbf{p}) = \sum_{\ell=1}^{d} K_\ell(q_\ell, p_\ell) \tag{11}$$

As a remark, additive kernels are mainly used for histograms. Therefore, we also follow existing work [53], [36], [49] and regard $\mathbf{q}$ and $\mathbf{p}$ as histograms, i.e., $q_\ell$ and $p_\ell$ are both positive in this section. Table 6 summarizes the most representative $K_\ell(q_\ell, p_\ell)$ [49], [53] for different additive kernels.

TABLE 6: $K_\ell(q_\ell, p_\ell)$ for different additive kernel functions

| Kernel | $K_\ell(q_\ell, p_\ell)$ |
|---|---|
| $\chi^2$ | $\frac{2q_\ell p_\ell}{q_\ell + p_\ell}$ |
| Intersection | $\min(q_\ell, p_\ell)$ |
| JS | $\frac{1}{2} q_\ell \log_2(\frac{q_\ell + p_\ell}{q_\ell}) + \frac{1}{2} p_\ell \log_2(\frac{q_\ell + p_\ell}{p_\ell})$ |
| Hellinger | $\sqrt{q_\ell p_\ell}$ |

Observe from Equation 11, additive kernel function $\mathcal{K}(\mathbf{q}, \mathbf{p})$ consists of different one-dimensional kernel functions $K_\ell(q_\ell, p_\ell)$. Therefore, we can convert the kernel aggregation function (cf. Equation 1) to the composition of one-dimensional kernel aggregation functions $\mathcal{F}_{P_\ell}(q_\ell)$ [36] (cf. Lemma 6), where $P_\ell$ denotes the set of data points in $P$ with the $\ell^{\text{th}}$ dimension, i.e., $P_\ell = \{p_{1\ell}, p_{2\ell}, ..., p_{n\ell}\}$, and $\mathcal{F}_{P_\ell}(q_\ell)$ is:

$$\mathcal{F}_{P_\ell}(q_\ell) = \sum_{p_{i\ell} \in P_\ell} w_i \cdot K_\ell(q_\ell, p_{i\ell}) \tag{12}$$

**Lemma 6.** *Given the additive kernel $\mathcal{K}(\mathbf{q}, \mathbf{p})$, the kernel aggregation function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1) is the addition of d one-dimensional kernel aggregation functions $\mathcal{F}_{P_\ell}(q_\ell)$, i.e.,*

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\ell=1}^{d} \mathcal{F}_{P_\ell}(q_\ell) \tag{13}$$

*Proof.*

$$\begin{aligned} \mathcal{F}_P(\mathbf{q}) &= \sum_{\mathbf{p_i} \in P} w_i \cdot \mathcal{K}(\mathbf{q}, \mathbf{p_i}) = \sum_{\mathbf{p_i} \in P} w_i \cdot \Big(\sum_{\ell=1}^{d} K_\ell(q_\ell, p_{i\ell})\Big) \\ &= \sum_{\ell=1}^{d} \Big(\sum_{p_{i\ell} \in P_\ell} w_i \cdot K_\ell(q_\ell, p_{i\ell})\Big) = \sum_{\ell=1}^{d} \mathcal{F}_{P_\ell}(q_\ell) \end{aligned}$$

□

The above lemma implies that we can focus on developing the lower and upper bound functions of one-dimensional kernel aggregation function $\mathcal{F}_{P_\ell}(q_\ell)$, and then add them to obtain the bounds of $\mathcal{F}_P(\mathbf{q})$.

In the following, we will present two bounding techniques for $\mathcal{F}_{P_\ell}(q_\ell)$, which are the monotonicity-based bounds (cf. Section 5.1) and the linear bounds (cf. Section 5.2). Then, we further propose a two-step method to integrate these two bounding techniques for solving $\epsilon$KAQ and $\tau$KAQ in Section 5.3.

### 5.1 Monotonicity-based Lower and Upper Bounds for $\mathcal{F}_{P_\ell}(q_\ell)$ (KARL$_{\text{mono}}$)

In this section, we develop the new lower and upper bound functions for $\mathcal{F}_{P_\ell}(q_\ell)$, which are based on the monotonicity property. To simplify the discussion, we mainly focus on the $\chi^2$ kernel function (cf. Table 6) and type-II weighting (i.e., $w_i \geq 0$). However, these bound functions can be extended to other additive kernel functions and other weighting (using the similar technique in Section 4.3.2).

Observe from Figure 7, the larger the $q_\ell$, the larger the function value $\frac{2q_\ell x}{q_\ell + x}$. Therefore, once we let $x = p_{i\ell}$ and we have computed these two values $\mathcal{F}_{\mathcal{P}_\ell}(\underline{q_\ell})$ and $\mathcal{F}_{\mathcal{P}_\ell}(\widehat{q_\ell})$, where $\underline{q_\ell} \leq q_\ell \leq \widehat{q_\ell}$. We can conclude $\mathcal{F}_{\mathcal{P}_\ell}(\underline{q_\ell}) \leq \mathcal{F}_{\mathcal{P}_\ell}(q_\ell) \leq \mathcal{F}_{\mathcal{P}_\ell}(\widehat{q_\ell})$, i.e., these two values act as the lower and upper bounds of $\mathcal{F}_{\mathcal{P}_\ell}(q_\ell)$ (cf. Lemma 7).

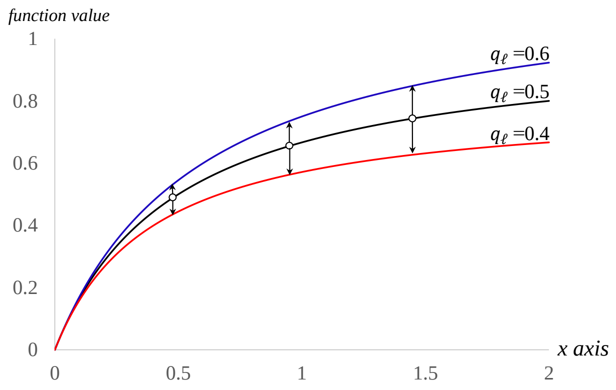


Fig. 7: Lower and upper bounds for $\frac{2q_\ell x}{q_\ell + x}$

**Lemma 7.** *If $\underline{q_\ell} \leq q_\ell \leq \widehat{q_\ell}$, we have $\mathcal{F}_{\mathcal{P}_\ell}(\underline{q_\ell}) \leq \mathcal{F}_{\mathcal{P}_\ell}(q_\ell) \leq \mathcal{F}_{\mathcal{P}_\ell}(\widehat{q_\ell})$, where the kernel functions are from Table 6.*

*Proof.* In this proof, we only focus on the $\chi^2$ kernel function. However, we can extend this proof to other kernel functions in Table 6 using the similar idea.

Given $q_\ell \leq \widehat{q_\ell}$, we have:

$$\begin{aligned} & \mathcal{F}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}_{P_\ell}(q_\ell) \\ = & \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2\widehat{q_\ell} p_{i\ell}}{\widehat{q_\ell} + p_{i\ell}} \Big) - \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2 q_\ell p_{i\ell}}{q_\ell + p_{i\ell}} \Big) \\ = & \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2(\widehat{q_\ell} - q_\ell) p_{i\ell}^2}{(q_\ell + p_{i\ell})(\widehat{q_\ell} + p_{i\ell})} \Big) \geq 0 \end{aligned}$$

Therefore, we have $\mathcal{F}_{P_\ell}(q_\ell) \leq \mathcal{F}_{P_\ell}(\widehat{q_\ell})$. Similarly, we can also conclude $\mathcal{F}_{\mathcal{P}_\ell}(\underline{q_\ell}) \leq \mathcal{F}_{\mathcal{P}_\ell}(q_\ell)$. □

Even though the bound functions $\mathcal{F}_{\mathcal{P}_\ell}(\underline{q_\ell})$ and $\mathcal{F}_{\mathcal{P}_\ell}(\widehat{q_\ell})$ can correctly act as the lower and upper bounds for $\mathcal{F}_{\mathcal{P}_\ell}(q_\ell)$ respectively, it is not feasible to compute these two bounds on-the-fly, since the response time is the same as the exact evaluation of $\mathcal{F}_{\mathcal{P}_\ell}(q_\ell)$. To avoid exact evaluation in the online phase, we precompute several $\mathcal{F}_{\mathcal{P}_\ell}(q)$ for different $q$ in advance, as shown in Figure 8. In the online phase, we can directly use the precomputed values, $\mathcal{F}_{\mathcal{P}_\ell}(\underline{q_\ell})$ and $\mathcal{F}_{\mathcal{P}_\ell}(\widehat{q_\ell})$ as the lower and upper bounds of $\mathcal{F}_{\mathcal{P}_\ell}(q_\ell)$ respectively, once $\underline{q_\ell} \leq q_\ell \leq \widehat{q_\ell}$.

Observe from Figure 8, there is a trade-off between the bound values and the precomputation cost (space and time). Here, we demonstrate how to sample the $q$-axis uniformly with interval size $\delta$ (cf. Figure 8) in offline phase, such that we can find the precomputed value to achieve the tolerance $\varepsilon$ between the lower bound $\mathcal{F}_{P_\ell}(\underline{q_\ell})$ and upper bound $\mathcal{F}_{P_\ell}(\widehat{q_\ell})$, i.e., $\mathcal{F}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}_{P_\ell}(\underline{q_\ell})$, which guarantees these two bounds are not far away from each other (cf. Lemma 8).

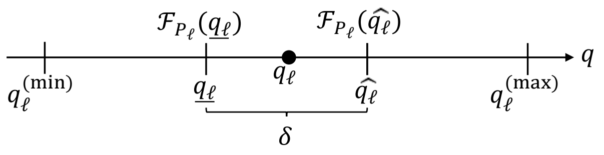


Fig. 8: Precomputation of $\mathcal{F}_{\mathcal{P}_\ell}(q)$ in the $\ell^{\text{th}}$ dimension (four $\mathcal{F}_{P_\ell}(q)$ are precomputed in this example)

**Lemma 8.** *Given $\delta$ as the interval size of consecutive samples, i.e., $\delta = \widehat{q_\ell} - \underline{q_\ell}$, and $\varepsilon$ as the tolerance, if $\delta$ fulfills the condition in Table 7, we conclude $\mathcal{F}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}_{P_\ell}(\underline{q_\ell})$.*

*Proof.* In this proof, we focus on the $\chi^2$ kernel function.

$$\begin{aligned} \mathcal{F}_{P_\ell}(\widehat{q_\ell}) &= \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2\widehat{q_\ell} p_{i\ell}}{\widehat{q_\ell} + p_{i\ell}} \Big) = \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2(\underline{q_\ell} + \delta) p_{i\ell}}{\underline{q_\ell} + \delta + p_{i\ell}} \Big) \\ &\leq \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2(\underline{q_\ell} + \delta) p_{i\ell}}{\underline{q_\ell} + p_{i\ell}} \Big) \\ &= \mathcal{F}_{P_\ell}(\underline{q_\ell}) + \delta \sum_{p_{i\ell} \in P_\ell} w_i \Big( \frac{2 p_{i\ell}}{\underline{q_\ell} + p_{i\ell}} \Big) \end{aligned}$$

Therefore, we have the following relative error:

$$\frac{\mathcal{F}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}_{P_\ell}(\underline{q_\ell})} \leq \frac{\delta}{\underline{q_\ell}} \leq \frac{\delta}{q_\ell^{(min)}}$$

To ensure the tolerance is within $\varepsilon$, we set:

$$\frac{\delta}{q_\ell^{(min)}} \leq \varepsilon \implies \delta \leq \varepsilon \times q_\ell^{(min)}$$

Hence, we prove the above claim. □

In the proof of Lemma 8, we only focus on the $\chi^2$ kernel. For other kernel functions in Table 6, we also have similar results, but with different conditions for $\delta$, which are summarized in Table 7. We denote $p_\ell^{(min)} = \min_{p_{i\ell} \in P_\ell} p_{i\ell}$ and $p_\ell^{(max)} = \max_{p_{i\ell} \in P_\ell} p_{i\ell}$. the detailed proofs for other kernel functions (intersection, JS and Hellinger) are shown in Appendix (cf. Section 8).

TABLE 7: Different conditions for interval size $\delta$

| Kernel | Condition for $\delta$ |
|---|---|
| $\chi^2$ | $\delta \leq \varepsilon \times q_\ell^{(min)}$ |
| Intersection | $\delta \leq \varepsilon \times \min(q_\ell^{(min)}, p_\ell^{(min)})$ |
| JS | $\delta \leq \varepsilon \times \min\Big(q_\ell^{(min)}, p_\ell^{(min)} \times \log_2\Big(\frac{q_\ell^{(min)}}{p_\ell^{(max)}} + 1\Big)\Big)$ |
| Hellinger | $\delta \leq \varepsilon^2 \times q_\ell^{(min)}$ |

If $q_\ell$ is in the range $[q_\ell^{(min)}, q_\ell^{(max)}]$, for every $\ell = 1, 2..., d$, we can obtain the lower and upper bounds, denoted as $\underline{F}$ and $\widehat{F}$ respectively, for $\mathcal{F}_P(\mathbf{q})$ in $O(d)$ time based on the lookup operations, where:

$$\underline{F} = \sum_{\ell=1}^{d} \mathcal{F}_{P_\ell}(\underline{q_\ell}) \text{ and } \widehat{F} = \sum_{\ell=1}^{d} \mathcal{F}_{P_\ell}(\widehat{q_\ell})$$

Moreover, these two bound values are also within the tolerance $\varepsilon$.

**Theorem 3.** *If $q_\ell$ is in the range of $[q_\ell^{(min)}, q_\ell^{(max)}]$ and the interval size $\delta$ fulfills the condition in Table 7, with $\varepsilon$ as the tolerance, for every $\ell = 1, 2..., d$, we have $\widehat{F} \leq (1+\varepsilon)\underline{F}$.*

*Proof.* If each $q_\ell$ is in the range of $[q_\ell^{(min)}, q_\ell^{(max)}]$, $q_\ell$ lies on one interval $[\underline{q_\ell}, \widehat{q_\ell}]$. Using the result in Lemma 8, $\mathcal{F}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}_{P_\ell}(\underline{q_\ell})$, for every $q_\ell \in [q_\ell^{(min)}, q_\ell^{(max)}]$. Therefore, we have:

$$\widehat{F} = \sum_{\ell=1}^{d} \mathcal{F}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon) \sum_{\ell=1}^{d} \mathcal{F}_{P_\ell}(\underline{q_\ell}) = (1+\varepsilon)\underline{F}$$

□

In both Lemma 8 and Theorem 3, there is one assumption that we need to know the range for $q_\ell$, i.e., $[q_\ell^{(min)}, q_\ell^{(max)}]$, in advance for each dimension. In practice, we can utilize the interval of each dimension in $P$ to estimate $[q_\ell^{(min)}, q_\ell^{(max)}]$, i.e., we estimate $q_\ell^{(min)} = p_\ell^{(min)}$ and $q_\ell^{(max)} = p_\ell^{(max)}$. Normally, $[p_\ell^{(min)}, p_\ell^{(max)}]$ is not large, as we need to normalize each dimension of the dataset to be within $[0, 1]$ before training the regression and classification models, e.g., SVM [12]. Note that the interval size $\delta$ depends on the value $q_\ell^{(min)}$ (cf. Lemma 8). We cannot set it to a very small value, e.g., 0, as it incurs very large precomputation cost, including space and time. As an example, if $q_\ell^{(min)} = 0.00001$ and $q_\ell^{(max)} = 1$ and $\varepsilon = 0.01$, then, for $\chi^2$ kernel, $\delta = 10^{-7}$ and the corresponding precomputation space (or the number of intervals) for each dimension is $\frac{(1-0.00001)}{10^{-7}}$, which is near 10 million. To avoid the huge precomputation cost, we restrict the smallest estimated value of $q_\ell^{(min)}$ to be 0.01, even though it is possible that $p_\ell^{(min)} < 0.01$.

### 5.2 Linear Bounds for Out-of-Range $q_\ell$ (KARL$_{\text{linear}}$)

As discussed in Section 5.1, we only utilize the interval of each dimension in $P$ to estimate $[q_\ell^{(min)}, q_\ell^{(max)}]$. However, it is possible for $q_\ell$ to be out-of-range in the online phase. To handle this situation, we extend our linear bounds (cf. Sections 4.1 and 4.2) for this case. As an example, we focus on the $\chi^2$ kernel in Equation 12 (i.e., Equation 14) in this section. However, our method can be also easily applied for other kernel functions.

$$\mathcal{F}_{P_\ell}(q_\ell) = \sum_{p_{i\ell} \in P_\ell} w_i \Big(\frac{2q_\ell p_{i\ell}}{q_\ell + p_{i\ell}}\Big) \tag{14}$$

In order to extend our linear bounds to Equation 14, we let $x_i = p_{i\ell}$ and define the following aggregation of linear function, where $Lin_{m,c}(x) = mx + c$.

$$\mathcal{FL}_{P_\ell}(q_\ell, Lin_{m,c}) = \sum_{p_{i\ell} \in P_\ell} w_i \cdot Lin_{m,c}(p_{i\ell})$$

Based on the similar concepts of Definition 2 and Lemma 1, our goal is to find two linear functions $Lin_{m_l,c_l}(x) = m_l x + c_l$ and $Lin_{m_u,c_u}(x) = m_u x + c_u$ such that $Lin_{m_l,c_l}(x) \leq \frac{2q_\ell x}{q_\ell + x} \leq Lin_{m_u,c_u}(x)$. Once we obtain these two linear functions, we have $\mathcal{FL}_{P_\ell}(q_\ell, Lin_{m_l,c_l}) \leq \mathcal{F}_{P_\ell}(q_\ell) \leq \mathcal{FL}_{P_\ell}(q_\ell, Lin_{m_u,c_u})$ (cf. Lemma 9).

**Lemma 9.** *Given $Lin_{m_l,c_l}(x) \leq \frac{2q_\ell x}{q_\ell + x} \leq Lin_{m_u,c_u}(x)$, we have: $\mathcal{FL}_{P_\ell}(q_\ell, Lin_{m_l,c_l}) \leq \mathcal{F}_{P_\ell}(q_\ell) \leq \mathcal{FL}_{P_\ell}(q_\ell, Lin_{m_u,c_u})$.*

Observe from Figure 9, since $\frac{2q_\ell x}{q_\ell + x}$ is the concave function, we can simply use the chord and tangent lines as $Lin_{m_l,c_l}(x)$ and $Lin_{m_u,c_u}(x)$ respectively. To find $m_l$, $c_l$, $m_u$ and $c_u$, we can utilize the similar concepts in Section 4.2, which are omitted here. The concave property can be also found for other additive kernel functions (cf. Table 6). Therefore, we can simply extend Lemma 9 for other additive kernel functions.

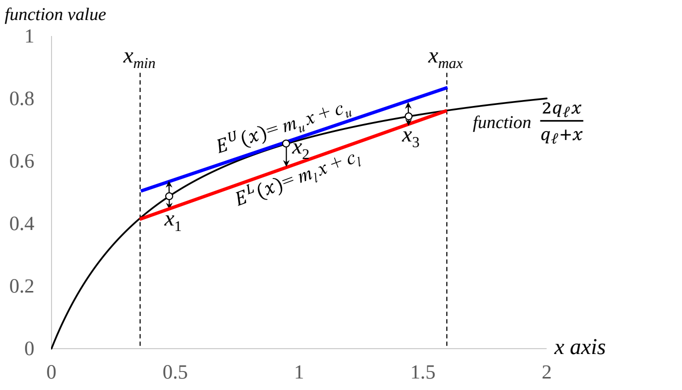


Fig. 9: Linear lower and upper bound functions for function $\frac{2q_\ell x}{q_\ell + x}$

After we find the suitable $Lin_{m,c}(x)$, we can obtain the $O(1)$-time bound function $\mathcal{FL}_{P_\ell}(q_\ell, Lin_{m,c})$ for $\mathcal{F}_{P_\ell}(q_\ell)$ (cf. Equation 14), as stated in Lemma 10.

**Lemma 10.** *Given two values $m$ and $c$, we can evaluate $\mathcal{FL}_{P_\ell}(q_\ell, Lin_{m,c})$ in $O(1)$ time.*

*Proof.*

$$\begin{aligned}\mathcal{FL}_{P_\ell}(q_\ell, Lin_{m,c}) &= \sum_{p_{i\ell} \in P_\ell} w_i \cdot Lin_{m,c}(p_{i\ell}) \\ &= \sum_{p_{i\ell} \in P_\ell} w_i \cdot (mp_{i\ell} + c) = ma_{P_\ell} + cb_P\end{aligned}$$

where $a_{P_\ell} = \sum_{p_{i\ell} \in P_\ell} w_i p_{i\ell}$ and $b_P = \sum_{p_{i\ell} \in P_\ell} w_i$ can be precomputed. □

To efficiently support the lower and upper bound functions for each dimension, we need to prebuild multiple index structures (e.g., multiple binary trees). Once $q_\ell$ is out-of-range, we can use the tree for $\ell^{th}$ dimension to find the approximate value $\widehat{F_\ell}$ which is within $\mathcal{F}_{P_\ell}(q_\ell)$ and $(1+\varepsilon)\mathcal{F}_{P_\ell}(q_\ell)$.

### 5.3 Integration of Our Bounds to Solve $\epsilon$KAQ and $\tau$KAQ (KARL)

In this section, we discuss how to integrate our bound functions (cf. Sections 5.1 and 5.2) to solve $\epsilon$KAQ and $\tau$KAQ.

Recall from Section 3, the termination conditions for $\tau$KAQ and $\epsilon$KAQ are $\widehat{lb} \geq \tau$ or $\widehat{ub} \leq \tau$ and $\widehat{ub} \leq (1+\epsilon)\widehat{lb}$ respectively. To solve $\epsilon$KAQ and $\tau$KAQ with additive kernels, we adopt the following 2-step method to maintain the bounds $\widehat{lb}$ and $\widehat{ub}$.

1) We either obtain the monotonicity-based lower and upper bound functions, i.e., $\mathcal{F}_{P_\ell}(\underline{q_\ell})$ and $\mathcal{F}_{P_\ell}(\widehat{q_\ell})$ respectively, (cf. Section 5.1) or compute the linear bounds once $q_\ell$ is out-of-range (cf. Section 5.2), using kd-tree/ ball-tree for $\ell^{\text{th}}$ dimensional points in $P$ (with the similar concept of Section 3). We denote the lower and upper bounds for $\ell^{\text{th}}$ dimension as $l_\ell$ and $u_\ell$ respectively. Then, we update $\widehat{lb}$ and $\widehat{ub}$.
2) If $\widehat{lb}$ and $\widehat{ub}$ does not fulfill the termination condition, we incrementally perform sequential scan (SCAN) for each

dimension (with the higher priority for larger value of $u_\ell - l_\ell$), i.e., compute $\mathcal{F}_{P_\ell}(q_\ell)$, to refine the bounds, $\widehat{lb}$ and $\widehat{ub}$, until the termination condition is fulfilled.

## 6 Experimental Evaluation

We introduce the experimental setting in Section 6.1. Next, we demonstrate the performance of different methods with Gaussian kernel function in Section 6.2. Lastly, we demonstrate the performance of different methods with $\chi^2$ kernel function in Sections 6.3 and 6.4.

### 6.1 Experimental Setting

#### *6.1.1 Datasets*

For type-I, type-II, type-III weighting, we take the application models as kernel density estimation, 1-class SVM, and 2-class SVM/SVR, respectively. We use a wide variety of real datasets for these models, as shown in Table 8. The value $n_{raw}$ denotes the number of points in the raw dataset, and the value $d$ denotes the data dimensionality. These datasets are all from data repository websites [2], [12]. Some datasets have been also used in existing work, e.g., [20], [41].

For type-I weighting, we follow [20] and use the Scott's rule to obtain the parameter $\gamma$. Type-II and type-III datasets require a training phase. We consider two kernel functions: Gaussian and $\chi^2$ kernels. We denote the number of remaining points in the dataset after training as $n^{gauss}_{model}$ and $n^{\chi^2}_{model}$, for the Gaussian kernel and $\chi^2$ kernel respectively.

The LIBSVM software [12] is used in the training phase. The training parameters are configured as follows. For each type-II dataset, we apply 1-class SVM training, with the default kernel parameter $\gamma = \frac{1}{d}$ [12]. Then we vary the training model parameter $\nu$ from 0.01 to 0.3 and choose the model which provides the highest accuracy. For each type-III dataset, we apply 2-class SVM/SVR training with the automatic script in [12] to determine the suitable values for training parameters.

TABLE 8: Details of datasets

| Model | Raw dataset | $n_{raw}$ | $n^{gauss}_{model}$ | $n^{\chi^2}_{model}$ | $d$ |
|---|---|---|---|---|---|
| Type I: | miniboone [2] | 119596 | n/a | n/a | 50 |
| kernel | home [2] | 918991 | n/a | n/a | 10 |
| density | susy [2] | 4990000 | n/a | n/a | 18 |
| Type II: | nsl-kdd [1] | 67343 | 17510 | n/a | 41 |
| 1-class | kdd99 [2] | 972780 | 19461 | n/a | 41 |
| SVM | covtype [12] | 581012 | 25486 | n/a | 54 |
| Type III: | cadata [2] | 10640 | 1643 | 1179 | 8 |
| 2-class | wave [2] | 62000 | 3454 | 2331 | 48 |
| SVR | 3D-road [2] | 424874 | 177585 | 176847 | 3 |
| Type III: | skin [2] | 235057 | 14628 | 28124 | 3 |
| 2-class | cod-rna [12] | 478565 | 114765 | 62368 | 8 |
| SVM | home [2] | 918991 | 411461 | 297560 | 10 |

#### *6.1.2 Methods for Comparisons*

In our experimental study, we compare different state-of-the-art methods with our solution. SCAN is the sequential scan method which computes $\mathcal{F}_P(\mathbf{q})$ without any pruning. SOTA is the state-of-the-art method which was developed by [20] for handling the kernel density classification problem, i.e., I-$\tau$ query. We modify and extend their framework to handle other types of queries. LIBSVM [12] is the famous library for handling both support vector machine-based regression and classification models, i.e., query types II-$\tau$, III-$\epsilon$ and III-$\tau$. In our preliminary version [10], we develop KARL for Gaussian kernel function, which follows the framework of [20], combining with our linear bound functions, $LB_{KARL}$ and $UB_{KARL}$. In this extension, we extend KARL to support additive kernels, we denote KARL$_{\text{mono}}$ as the method of monotonicity-based bound functions (cf. Section 5.1) and KARL$_{\text{linear}}$ as the method of linear bound functions (cf. Section 5.2) with the combination of multiple binary-trees (for each dimension). We further integrate both methods KARL$_{\text{mono}}$ and KARL$_{\text{linear}}$ (which handles out-of-range case of $q_\ell$) to an unified one (cf. Section 5.3), i.e., KARL, to support the $\chi^2$ kernel function for both query types III-$\epsilon$ and III-$\tau$.

### 6.2 Efficiency Evaluation for Different Query Types with Gaussian kernel

We test the performance of different methods for five types of queries which are I-$\epsilon$, I-$\tau$, II-$\tau$, III-$\epsilon$ and III-$\tau$. The parameters of these queries are set as follows.

**Types I-$\epsilon$ and III-$\epsilon$.** We set the relative error $\epsilon = 0.2$ for each dataset.

**Type I-$\tau$.** We fix the mean value $\mu$ of $\mathcal{F}_P(\mathbf{q})$ from the set $Q$, i.e., $\mu = \sum_{\mathbf{q}\in Q} \mathcal{F}_P(\mathbf{q})/|Q|$ as the threshold $\tau$ for each dataset in Table 9.

**Types II-$\tau$ and III-$\tau$.** The threshold $\tau$ can be obtained during the training phase.

TABLE 9: Throughput (queries/sec) of all methods for different types of queries with Gaussian kernel

| Type | Datasets | SCAN | LIBSVM | Scikit | SOTA | KARL |
|---|---|---|---|---|---|---|
| I-$\epsilon$ | miniboone | 36.1 | n/a | 36 | 16.5 | **301** |
| | home | 15.2 | n/a | 11.9 | 36.2 | **187** |
| | susy | 2.02 | n/a | 1.17 | 0.77 | **13.2** |
| I-$\tau$ | miniboone | 36.1 | n/a | n/a | 102 | **510** |
| | home | 15.2 | n/a | n/a | 93.2 | **258** |
| | susy | 2.02 | n/a | n/a | 3.58 | **83.4** |
| II-$\tau$ | nsl-kdd | 283 | 481 | n/a | 748 | **20668** |
| | kdd99 | 260 | 520 | n/a | 1269 | **11324** |
| | covtype | 158 | 462 | n/a | 448 | **6022** |
| III-$\epsilon$ | cadata | 1951 | 2001 | n/a | 1756 | **13539** |
| | wave | 940 | 1205 | n/a | 442 | **33482** |
| | 3D-road | 41 | 62 | n/a | 28 | **27334** |
| III-$\tau$ | skin | 495 | 504 | n/a | 248 | **4104** |
| | cod-rna | 55.4 | 68.2 | n/a | 27.6 | **854** |
| | home | 14.7 | 18.3 | n/a | 8.6 | **231** |

Table 9 shows the throughput of different methods for all types of queries. In the result of query type I-$\epsilon$, SCAN is comparable to Scikit and SOTA since the bounds computed by the basic bound functions are not tight enough. The performance of Scikit and SOTA is affected by the overhead of the loose bound computations. KARL uses our new bound functions which are shown to provide tighter bounds. These bounds lead to significant speedup in all evaluated datasets, e.g., KARL is at least 5.16 times faster than other methods. For query type III-$\epsilon$, our method KARL can further improve the efficiency performance by 7x to 28x, compared with other methods.

For query type I-$\tau$, our method KARL improves the throughput by 2.76x to 21.2x when comparing to the runner-up method SOTA. The improvement becomes more obvious for type II-$\tau$ and type III-$\tau$ queries. The improvement of KARL can be up to 31x as compared to SOTA. KARL achieves significant performance gain for all these queries due to its tighter bound value compared with SOTA.

**Sensitivity of $\tau$.** In order to test the sensitivity of threshold $\tau$ in different methods, we select five thresholds from the range $\mu -$

$\sigma$ to $\mu + 3\sigma$, where $\sigma = \sqrt{\sum_{\mathbf{q}\in Q}(\mathcal{F}_P(\mathbf{q}) - \mu)^2/|Q|}$ is the standard deviation. Figure 10 shows the results on the two largest datasets (home and susy). Due to the superior performance of our bound functions, KARL outperforms SOTA by nearly one order of magnitude in most of the datasets regardless of the chosen threshold.

**Sensitivity of $\epsilon$.** In Scikit-learn library [40], we can select different relative error $\epsilon$ in the approximate KDE. To test the sensitivity, we vary the relative error $\epsilon$ for the two largest datasets (home and susy) with query type I-$\epsilon$. If the value of $\epsilon$ is very small, the room for the bound estimations is very limited so that neither KARL nor SOTA perform well in very small $\epsilon$ setting (e.g., 0.05). For other general $\epsilon$ settings, our method KARL consistently outperforms other methods by a visible margin (cf. Figure 11).

**Sensitivity of dataset size.** In the following experiment, we test how the size of the dataset affects the evaluation performance of different methods for both query types I-$\epsilon$ and I-$\tau$. We choose the largest dataset (susy) and vary the size via sampling. The trend in Figure 12 meets our expectation; a smaller size implies a higher throughput. Our KARL in general outperforms other methods by one order of magnitude in a wide range of data sizes.

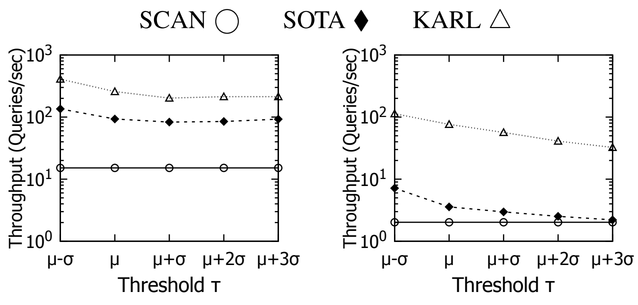
## 6.3 Efficiency Evaluation for Different Query Types with $\chi^2$ kernel

In this section, we test the efficiency performance for $\chi^2$ kernel (cf. Table 10). However, our techniques can be extended to other additive kernel functions. Since additive kernels are mainly used in regression (query type III-$\epsilon$) and classification (query type III-$\tau$) models, we only conduct the experiments in query type III. By default, we fix the relative error $\epsilon = 0.2$ for $\epsilon$KAQ. In addition, we also fix the tolerance parameter $\varepsilon = 0.05$ for the KARL$_{\text{mono}}$ method. Since we fix $q_\ell^{(min)} = 0.01$ (cf. Section 5.1), we can choose the interval size $\delta = \varepsilon \times q_\ell^{(min)} = 0.0005$ ($\chi^2$ kernel in Table 7). As such, the number of precomputed $q_\ell$ is $\frac{1-0.01}{0.0005} = 1980$. Observe from Table 10, both our method KARL$_{\text{mono}}$ and our method KARL can be significantly better than the existing methods (e.g., SCAN, LibSVM and SOTA) by at least one order of magnitude. On the other hand, since the monotonicity-based bound functions are tighter than linear bound functions, KARL$_{\text{mono}}$ can also achieve significant speedup compared with our method KARL$_{\text{linear}}$ [10].

TABLE 10: Throughput (queries/sec) of all methods for different types of queries with $\chi^2$ kernel

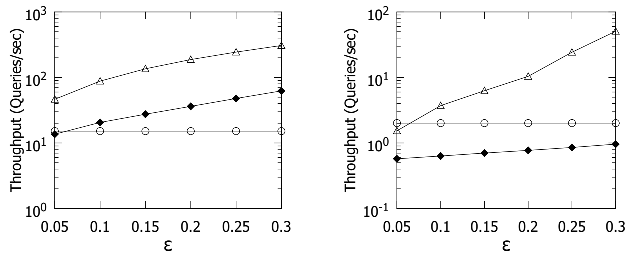
| Type | Datasets | SCAN | LIBSVM | SOTA | KARL$_{\text{linear}}$ | KARL$_{\text{mono}}$ | KARL |
|---|---|---|---|---|---|---|---|
| III-$\epsilon$ | cadata | 941 | 1102 | 366 | 1438 | 13243 | **15684** |
| | wave | 87 | 104 | 26.8 | 149 | 2863 | **3216** |
| | 3D-road | 18.1 | 26.3 | 3.7 | 45 | 183 | **201** |
| III-$\tau$ | skin | 115 | 180 | 303 | 6741 | 23273 | **31232** |
| | cod-rna | 24.3 | 32.5 | 52.5 | 701 | 10187 | **14875** |
| | home | 4.25 | 7.73 | 6.92 | 824 | 1919 | **2457** |

To test the sensitivity of the relative error $\epsilon$ for $\epsilon$KAQ with $\chi^2$ kernel function, we vary the parameter $\epsilon$ from 0.05 to 0.3, and measure the throughput of each method, using the two largest datasets wave and 3D-road. Observe from Figure 13, since our monotonicity-based lower and upper bound functions are effective, both our method KARL$_{\text{mono}}$ and KARL can outperform the existing methods by at least one order of magnitude.



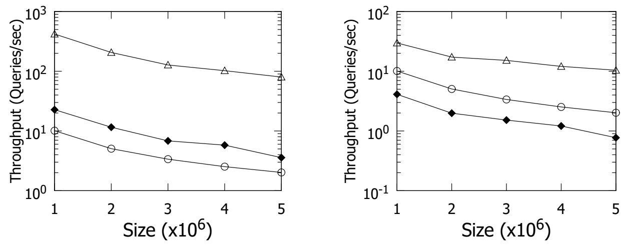
(a) home (b) susy

Fig. 10: Query throughput with query type I-$\tau$, varying the threshold $\tau$



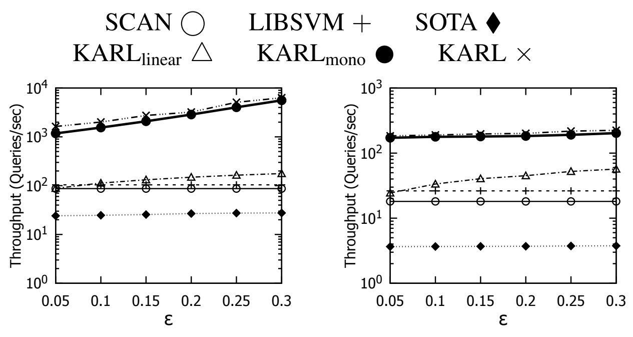
(a) home (b) susy

Fig. 11: Query throughput with query type I-$\epsilon$, varying the relative error $\epsilon$



(a) type I-$\tau$, fixing $\tau = \mu$ (b) type I-$\epsilon$, fixing $\epsilon = 0.2$

Fig. 12: Query throughput on the susy dataset, varying the dataset size



(a) wave (b) 3D-road

Fig. 13: Query throughput on $\chi^2$ kernel, varying the relative error $\epsilon$

## 6.4 Accuracy Evaluation for Different Methods with $\chi^2$ kernel

As discussed in Section 2, there are many approximation methods for supporting fast evaluation of kernel aggregation function using additive kernels. However, all of these methods do not provide theoretical guarantee between the returned result and the exact solution for both $\tau$KAQ and $\epsilon$KAQ. In this section, we compare the accuracy of three state-of-the-art approximation methods in both computer vision and machine learning communities, which are $\text{NN}_{\text{map}}$ [53], $\text{PPL}_{\text{map}}$ [41] and $\text{VLFeat}_{\text{map}}$ [49], and the exact methods (e.g., SCAN, KARL, etc.) in the classification task (i.e., $\tau$KAQ). Both $\text{NN}_{\text{map}}$, $\text{PPL}_{\text{map}}$ and $\text{VLFeat}_{\text{map}}$ create the high-dimensional feature maps (or feature vectors), which can be further trained/ predicted by the linear SVM [18] efficiently, to approximate the additive kernel functions. However, these approximation methods normally sacrifice the accuracy performance, compared with exact methods (e.g., KARL). Table 11 summarizes the accuracy result for $\chi^2$ kernel function. The accuracy of both $\text{VLFeat}_{\text{map}}$, $\text{PPL}_{\text{map}}$ and $\text{NN}_{\text{map}}$ are 8.6%-14.4%, 7.32%-8.53% and 5.36%-17.1% lower than exact methods respectively.

TABLE 11: Accuracy result (in %) for different methods with $\chi^2$ kernel

| Datasets | Exact (e.g., KARL) | $\text{VLFeat}_{\text{map}}$ [49] | $\text{PPL}_{\text{map}}$ [41] | $\text{NN}_{\text{map}}$ [53] |
|---|---|---|---|---|
| skin | **97.57** | 83.13 | 90.25 | 92.21 |
| cod-rna | **96.69** | 86.22 | 88.35 | 79.63 |
| home | **87.36** | 78.75 | 78.83 | 75.24 |

## 7 Conclusion

In this paper, we propose the solution, called KARL, to support the kernel aggregation queries, which are used in different types of machine learning models, including kernel density estimation/classification [21], [20], 1-class SVM [37], 2-class SVM and SVR [47].

Our contribution is threefold. First, we extend our work [10] to support other important kernel functions, called additive kernels, which are widely used in different communities, e.g., machine learning, computer vision, Geoscience, etc. Then, we develop the monotonicity-based bound functions for these kernel functions. After that, we further integrate both monotonicity-based bound functions and the linear bound functions to one unified solution to support these kernel functions. Experimental studies on a wide variety of datasets show that our solution KARL yields higher throughput than the state-of-the-art solution by at least one order of magnitude for kernel aggregation queries with additive kernel functions.

In the future, we will develop efficient algorithms for the prediction phase of other machine learning models, e.g., kernel clustering [52], graph-kernel-based classification [51] and kernelized support tensor machines [22]. On the other hand, we will also extend our work to the training phase of kernel-based machine learning models.

## Acknowledgement

This work was supported by the National Key Research and Development Plan of China (No.2019YFB2102100). Tsz Nam Chan and Reynold Cheng were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116, 106150091, and 17205115), the University of Hong Kong (Projects 104004572, 102009508, and 104004129), and the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18). Leong Hou U was supported by the Science and Technology Development Fund, Macau SAR (File no. 0015/2019/AKP and SKL-IOTSC-2018-2020) and University of Macau (File no. MYRG2019-00119-FST). Man Lung Yiu was supported by grant GRF 152050/19E from the Hong Kong RGC.

## References

[1] Nsl-kdd dataset. https://github.com/defcom17/.
[2] UCI machine learning repository. http://archive.ics.uci.edu/ml/index.php.
[3] J. Baek, S. Hong, J. Kim, and E. Kim. Efficient pedestrian detection at nighttime using a thermal camera. *Sensors*, 17(8):1850, 2017.
[4] J. Baek, J. Hyun, and E. Kim. A pedestrian detection system accelerated by kernelized proposals. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13, 2019.
[5] J. Baek, J. Kim, and E. Kim. Fast and efficient pedestrian detection via the cascade implementation of an additive kernel support vector machine. *IEEE Trans. Intelligent Transportation Systems*, 18(4):902–916, 2017.
[6] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys and Tutorials*, 16(1):303–336, 2014.
[7] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys and Tutorials*, 18(2):1153–1176, 2016.
[8] T. N. Chan, M. L. Yiu, and K. A. Hua. A progressive approach for similarity search on matrix. In *SSTD*, pages 373–390. Springer, 2015.
[9] T. N. Chan, M. L. Yiu, and K. A. Hua. Efficient sub-window nearest neighbor search on matrix. *IEEE Trans. Knowl. Data Eng.*, 29(4):784–797, 2017.
[10] T. N. Chan, M. L. Yiu, and L. H. U. KARL: fast kernel aggregation queries. In *ICDE*, pages 542–553, 2019.
[11] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.
[12] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[13] Q. Chen, Z. Song, J. Dong, Z. Huang, Y. Hua, and S. Yan. Contextualizing object detection and classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(1):13–27, 2015.
[14] H.-S. Chiu and et al. Pan-Cancer Analysis of lncRNA Regulation Supports Their Targeting of Cancer Genes in Each Tumor Context. *Cell Reports*, 23(1):297–312, Apr. 2018.
[15] K. Cranmer. Kernel estimation in high-energy physics. 136:198–207, 2001.
[16] B. Demir and L. Bruzzone. Histogram-based attribute profiles for classification of very high resolution remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(4):2096–2107, April 2016.
[17] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen. Additive gaussian processes. In *NIPS*, pages 226–234, 2011.
[18] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.
[19] B. J. Ferdosi, H. Buddelmeijer, S. C. Trager, M. H. F. Wilkinson, and J. B. T. M. Roerdink. Comparison of density estimation methods for astronomical datasets. *Astronomy and Astrophysics*, 2011.
[20] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.
[21] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.
[22] L. He, C. Lu, G. Ma, S. Wang, L. Shen, P. S. Yu, and A. B. Ragin. Kernelized support tensor machines. In *ICML*, pages 1442–1451, 2017.
[23] M. Herbster. Learning additive models online with fast evaluating kernels. In *COLT*, pages 444–460, 2001.
[24] C. Hsieh, S. Si, and I. S. Dhillon. Fast prediction for large-scale kernel machines. In *NIPS*, pages 3689–3697, 2014.
[25] C. Huang, G. Min, Y. Wu, Y. Ying, K. Pei, and Z. Xiang. Time series anomaly detection for trustworthy services in cloud computing systems. *IEEE Trans. Big Data*, 2017.
[26] P. Isola, J. Xiao, D. Parikh, A. Torralba, and A. Oliva. What makes a photograph memorable? *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(7):1469–1482, 2014.
[27] H. G. Jung and G. Kim. Support vector number reduction: Survey and experimental evaluations. *IEEE Trans. Intelligent Transportation Systems*, 15(2):463–476, 2014.
[28] A. Kampouraki, G. Manis, and C. Nikou. Heartbeat time series classification with support vector machines. *IEEE Trans. Information Technology in Biomedicine*, 13(4):512–518, 2009.
[29] Q. V. Le, T. Sarlós, and A. J. Smola. Fastfood - computing hilbert space expansions in loglinear time. In *ICML*, pages 244–252, 2013.
[30] W. Li, M. Coats, J. Zhang, and S. Mckenna. Discriminating dysplasia: Optical tomographic texture analysis of colorectal polyps. *Medical image analysis*, 26:57–69, 09 2015.

[31] B. Liu, Y. Xiao, P. S. Yu, L. Cao, Y. Zhang, and Z. Hao. Uncertain one-class learning and concept summarization learning on uncertain data streams. *IEEE Trans. Knowl. Data Eng.*, 26(2):468–484, 2014.

[32] Y. Liu, Y. Liu, and Y. Chen. Fast support vector data descriptions for novelty detection. *IEEE Trans. Neural Networks*, 21(8):1296–1313, 2010.

[33] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *IJCNN*, pages 1741–1745, 2003.

[34] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *ICCV*, pages 40–47, 2009.

[35] S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR*, 2008.

[36] S. Maji, A. C. Berg, and J. Malik. Efficient classification for additive kernel svms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):66–77, 2013.

[37] L. M. Manevitz and M. Yousef. One-class svms for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.

[38] A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, pages 397–405, 2000.

[39] M. Mutny and A. Krause. Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In *NeurIPS*, pages 9019–9030, 2018.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] O. Pele, B. Taskar, A. Globerson, and M. Werman. The pairwise piecewise-linear embedding for efficient non-linear classification. In *ICML*, pages 205–213, 2013.

[42] Y. Ren, P. N. Suganthan, and N. Srikanth. A novel empirical mode decomposition with support vector regression for wind speed forecasting. *IEEE Trans. Neural Netw. Learning Syst.*, 27(8):1793–1798, 2016.

[43] D. Sahoo, S. C. H. Hoi, and B. Li. Online multiple kernel regression. In *SIGKDD*, pages 293–302, 2014.

[44] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann. 2006.

[45] G. Santamaría-Bonfil, A. Reyes-Ballesteros, and C. Gershenson. Wind speed forecasting for wind farms: A method based on support vector regression. *Renewable Energy*, 85(C):790–809, 2016.

[46] N. I. Sapankevych and R. Sankar. Time series prediction using support vector machines: A survey. *IEEE Comp. Int. Mag.*, 4(2):24–38, 2009.

[47] B. Schölkopf and A. J. Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning series. MIT Press, 2002.

[48] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *NIPS*, pages 582–588, 1999.

[49] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):480–492, 2012.

[50] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *CVPR*, pages 2320–2327, 2012.

[51] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, 2010.

[52] S. Wang, A. Gittens, and M. W. Mahoney. Scalable kernel k-means clustering with nystr\"om approximation: Relative-error bounds. *J. Mach. Learn. Res.*, 20:12:1–12:49, 2019.

[53] Z. Wang, X. Yuan, Q. Liu, and S. Yan. Additive nearest neighbor feature maps. In *ICCV*, pages 2866–2874, 2015.

[54] K. Were, D. T. Bui, Øystein B. Dick, and B. R. Singh. A comparative assessment of support vector regression, artificial neural networks, and random forests for predicting and mapping soil organic carbon stocks across an afromontane landscape. *Ecological Indicators*, 52:394 – 403, 2015.

[55] B. Wolff, J. Kühnert, E. Lorenz, O. Kramer, and D. Heinemann. Comparing support vector regression for pv power forecasting to a physical modeling approach using measurement, numerical weather prediction, and cloud motion data. *Solar Energy*, 135(C):197–208, 2016.

[56] J. Wu, W. Tan, and J. M. Rehg. Efficient and effective visual codebook generation using additive kernels. *Journal of Machine Learning Research*, 12:3097–3118, 2011.

[57] L. Zhang, J. Lin, and R. Karim. Adaptive kernel density-based anomaly detection for nonlinear systems. *Knowledge-Based Systems*, 139(Supplement C):50 – 63, 2018.

[58] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*, pages 433–444, 2013.

**Tsz Nam Chan** received the bachelor's degree in electronic and information engineering and the PhD degree in computer science from the Hong Kong Polytechnic University in 2014 and 2019 respectively. He is currently a research associate in the University of Hong Kong. His research interests include multidimensional similarity search, pattern matching and kernel methods for machine learning.

**Leong Hou U** completed his B.Sc. in Computer Science and Information Engineering at Taiwan Chi Nan University, his M.Sc. in E-commerce at University of Macau, and his Ph.D. in Computer Science at University of Hong Kong. He is now an Associate Professor in the State Key Laboratory of Internet of Things for Smart City and the Department of Computer and Information Science, University of Macau. His research interests include spatial and spatio-temporal databases, advanced query processing, crowdsourced query processing, information retrieval, data mining and optimization problems.

**Reynold Cheng** is an Associate Professor of the Department of Computer Science in the University of Hong Kong (HKU). He obtained his PhD from Department of Computer Science of Purdue University in 2005. Dr. Cheng was granted an Outstanding Young Researcher Award 2011-12 by HKU.

**Man Lung Yiu** received the bachelor's degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an associate professor in the Department of Computing, The Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular query processing topics on spatiotemporal data and multidimensional data.

**Shivansh Mittal** is currently a final year undergraduate student, studying computer science major and finance minor, at the University of Hong Kong. His research interests include kernel methods for machine learning and recommender systems.

## 8 Appendix

In this appendix, we provide the detailed proofs in Lemma 8 (with different conditions of $\delta$ in Table 7) for other kernel functions, including both Intersection, JS and Hellinger kernels.

### 8.1 Proof of Lemma 8 with Intersection kernel

*Proof.*

$$\begin{aligned}
&\mathcal{F}_{P_\ell}(\widehat{q_\ell})\\
=\ & \sum_{p_{i\ell}\in P_\ell} w_i \min(\widehat{q_\ell}, p_{i\ell}) = \sum_{p_{i\ell}\in P_\ell} w_i \min(\underline{q_\ell}+\delta, p_{i\ell})\\
\leq\ & \sum_{p_{i\ell}\in P_\ell} w_i (\min(\underline{q_\ell}, p_{i\ell})+\delta) = \mathcal{F}_{P_\ell}(\underline{q_\ell}) + \delta \sum_{p_{i\ell}\in P_\ell} w_i
\end{aligned}$$

Therefore, we have the relative error:

$$\begin{aligned}
\frac{\mathcal{F}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}_{P_\ell}(\underline{q_\ell})} &\leq \frac{\delta \sum_{p_{i\ell}\in P_\ell} w_i}{\sum_{p_{i\ell}\in P_\ell} w_i \min(\underline{q_\ell}, p_{i\ell})}\\
&\leq \frac{\delta}{\min(q_\ell^{(\min)}, p_\ell^{(\min)})}
\end{aligned}$$

Once we set $\delta \leq \varepsilon \times \min(q_\ell^{(\min)}, p_\ell^{(\min)})$, we can achieve the tolerance $\varepsilon$, i.e., $\mathcal{F}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}_{P_\ell}(\underline{q_\ell})$. □

### 8.2 Proof of Lemma 8 with JS kernel

*Proof.* To prove this lemma, we separate $\mathcal{F}_{P_\ell}(q_\ell)$ into two parts, which are $\mathcal{F}^{(1)}_{P_\ell}(q_\ell)$ and $\mathcal{F}^{(2)}_{P_\ell}(q_\ell)$.

$$\begin{aligned}
&\mathcal{F}_{P_\ell}(q_\ell)\\
&= \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} q_\ell \log_2\Big(\frac{q_\ell + p_{i\ell}}{q_\ell}\Big) + \frac{1}{2} p_\ell \log_2\Big(\frac{q_\ell + p_{i\ell}}{p_{i\ell}}\Big)\Big)\\
&= \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} q_\ell \log_2\Big(\frac{q_\ell + p_{i\ell}}{q_\ell}\Big)\Big) + \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} p_{i\ell} \log_2\Big(\frac{q_\ell + p_{i\ell}}{p_{i\ell}}\Big)\Big)\\
&= \mathcal{F}^{(1)}_{P_\ell}(q_\ell) + \mathcal{F}^{(2)}_{P_\ell}(q_\ell)
\end{aligned}$$

Then, we claim that (1) $\mathcal{F}^{(1)}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell})$ and (2) $\mathcal{F}^{(2)}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})$, once $\delta$ fulfills the condition (cf. Table 7). After we prove the above claims, we can conclude $\mathcal{F}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}_{P_\ell}(\underline{q_\ell})$, since:

$$\begin{aligned}
&\frac{\mathcal{F}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}_{P_\ell}(\underline{q_\ell})}\\
=\ &\frac{\mathcal{F}^{(1)}_{P_\ell}(\widehat{q_\ell}) + \mathcal{F}^{(2)}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell}) - \mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell}) + \mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})} \leq \varepsilon
\end{aligned}$$

Now, we prove the above claims.

**Claim (1):**

$$\begin{aligned}
\mathcal{F}^{(1)}_{P_\ell}(\widehat{q_\ell}) &= \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2}\widehat{q_\ell} \log_2\Big(\frac{\widehat{q_\ell} + p_{i\ell}}{\widehat{q_\ell}}\Big)\Big)\\
&= \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2}(\underline{q_\ell}+\delta) \log_2\Big(\frac{\underline{q_\ell} + \delta + p_{i\ell}}{\underline{q_\ell}+\delta}\Big)\Big)\\
&\leq \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2}(\underline{q_\ell}+\delta) \log_2\Big(1 + \frac{p_{i\ell}}{\underline{q_\ell}}\Big)\Big)\\
&= \mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell}) + \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2}\delta \log_2\Big(\frac{\underline{q_\ell} + p_{i\ell}}{\underline{q_\ell}}\Big)\Big)
\end{aligned}$$

Therefore, we have:

$$\frac{\mathcal{F}^{(1)}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell})} = \frac{\sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2}\delta \log_2\Big(\frac{\underline{q_\ell}+p_{i\ell}}{\underline{q_\ell}}\Big)\Big)}{\sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2}\underline{q_\ell} \log_2\Big(\frac{\underline{q_\ell}+p_{i\ell}}{\underline{q_\ell}}\Big)\Big)} \leq \frac{\delta}{q_\ell^{(min)}}$$

Therefore, once we set $\delta \leq \varepsilon \times q_\ell^{(min)}$, we can achieve the tolerance $\varepsilon$ for the first part, where $\mathcal{F}^{(1)}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}^{(1)}_{P_\ell}(\underline{q_\ell})$.

**Claim (2):**

$$\begin{aligned}
\mathcal{F}^{(2)}_{P_\ell}(\widehat{q_\ell}) &= \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} p_{i\ell} \log_2\Big(\frac{\widehat{q_\ell} + p_{i\ell}}{p_{i\ell}}\Big)\Big)\\
&= \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} p_{i\ell} \log_2\Big(\frac{\underline{q_\ell} + \delta + p_{i\ell}}{p_{i\ell}}\Big)\Big)\\
&\leq \sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} p_{i\ell} \Big(\log_2\Big(\frac{\underline{q_\ell} + p_{i\ell}}{p_{i\ell}}\Big) + \frac{\delta}{p_{i\ell}}\Big)\Big)\\
&= \mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell}) + \frac{\delta}{2} \sum_{p_{i\ell}\in P_\ell} w_i
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}
\frac{\mathcal{F}^{(2)}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})} &\leq \frac{\frac{\delta}{2}\sum_{p_{i\ell}\in P_\ell} w_i}{\sum_{p_{i\ell}\in P_\ell} w_i \Big(\frac{1}{2} p_{i\ell} \log_2\Big(\frac{\widehat{q_\ell}+p_{i\ell}}{p_{i\ell}}\Big)\Big)}\\
&\leq \frac{\delta}{p_\ell^{(min)} \log\Big(\frac{q_\ell^{(min)}}{p_\ell^{(max)}} + 1\Big)}
\end{aligned}$$

As such, once we set $\delta \leq \varepsilon \times p_\ell^{(min)} \log\Big(\frac{q_\ell^{(min)}}{p_\ell^{(max)}} + 1\Big)$, $\mathcal{F}^{(2)}_{P_\ell}(\widehat{q_\ell})$ can achieve the tolerance $\varepsilon$ compared with $\mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})$, i.e., $\mathcal{F}^{(2)}_{P_\ell}(\widehat{q_\ell}) \leq (1+\varepsilon)\mathcal{F}^{(2)}_{P_\ell}(\underline{q_\ell})$.

To fulfill both claims of (1) and (2), we select the smallest $\delta$, i.e., $\delta = \varepsilon \times \min\Big(q_\ell^{(min)}, p_\ell^{(min)} \times \log_2\Big(\frac{q_\ell^{(min)}}{p_\ell^{(max)}} + 1\Big)\Big)$. □

### 8.3 Proof of Lemma 8 with Hellinger kernel

*Proof.*

$$\begin{aligned}
\mathcal{F}_{P_\ell}(\widehat{q_\ell}) &= \sum_{p_{i\ell}\in P_\ell} w_i \sqrt{\widehat{q_\ell} p_{i\ell}} = \sum_{p_{i\ell}\in P_\ell} w_i \sqrt{(\underline{q_\ell}+\delta) p_{i\ell}}\\
&\leq \sum_{p_{i\ell}\in P_\ell} w_i (\sqrt{\underline{q_\ell} p_{i\ell}} + \sqrt{\delta p_{i\ell}})\\
&= \mathcal{F}_{P_\ell}(\underline{q_\ell}) + \sum_{p_{i\ell}\in P_\ell} w_i \sqrt{\delta p_{i\ell}}
\end{aligned}$$

Therefore, we have the relative error:

$$\frac{\mathcal{F}_{P_\ell}(\widehat{q_\ell}) - \mathcal{F}_{P_\ell}(\underline{q_\ell})}{\mathcal{F}_{P_\ell}(\underline{q_\ell})} \leq \frac{\sum_{p_{i\ell}\in P_\ell} w_i \sqrt{\delta p_{i\ell}}}{\sum_{p_{i\ell}\in P_\ell} w_i \sqrt{\underline{q_\ell} p_{i\ell}}} = \sqrt{\frac{\delta}{q_\ell^{(min)}}}$$

Therefore, we can achieve the tolerance $\varepsilon$ once we set $\sqrt{\frac{\delta}{q_\ell^{(min)}}} \leq \varepsilon$ and thus, $\delta \leq \varepsilon^2 \times q_\ell^{(min)}$. □