KARL: Fast Kernel Aggregation Queries

Tsz Nam Chan Man Lung Yiu
Department of Computing
Hong Kong Polytechnic University
{cstnchan,csmlyiu}@comp.polyu.edu.hk

Leong Hou U
Department of Computing and Information Science
University of Macau
ryanlhu@umac.mo

***Abstract*—Kernel functions support a broad range of applications that require tasks like density estimation, classification, or outlier detection. In these tasks, a common online operation is to compute the weighted aggregation of kernel function values with respect to a set of points. Scalable aggregation methods are still unknown for typical kernel functions (e.g., Gaussian kernel, polynomial kernel, and sigmoid kernel) and weighting schemes. In this paper, we propose a novel and effective bounding technique to speedup the computation of kernel aggregation. We further boost its efficiency by leveraging index structures and exploiting index tuning opportunities. In addition, our technique is extensible to different types of kernel functions and weightings. Experimental studies on many real datasets reveal that our proposed method achieves speedups of 2.5–800 over the state-of-the-art.**

I. Introduction

In this era of digitalization, vast amount of data are being continuously collected and analyzed. Kernel functions are typically used in two tasks: (i) kernel density estimation (for statistical analysis) and (ii) support vector machine classification (for data mining). These tasks are actively used in the following applications. Network security systems [5], [4] utilize kernel SVM to detect suspicious packets. In medical science, medical scientists [9] utilize kernel SVM to identify tumor samples. Astronomical scientists [3] utilize kernel density estimation for quantifying the galaxy density. In particle physics, physicists utilize kernel density estimation to search for particles [10]. For example, Figure 1 illustrates the usage of kernel density estimation on a real dataset (miniboone [2]) for searching particles. Physicists are interested in the dense region (in yellow).

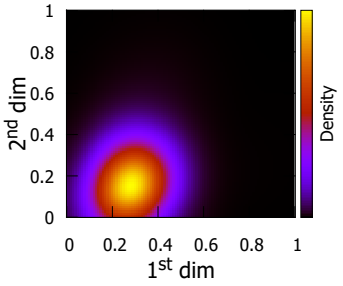


Fig. 1: Kernel density estimation on the miniboone dataset, using 1^{st} and 2^{nd} dimensions

Implementation-wise, both commercial database systems (e.g., Oracle, Vertica) and open-source libraries (e.g., LibSVM [7]) provide functions for support vector machines (SVM), which can combine with different kernel functions.

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

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) \quad (1)$$

where $\mathbf{q}$ is a query point, P is a dataset of points, w_i, γ are scalars, and $dist(\mathbf{q}, \mathbf{p_i})$ denotes the Euclidean distance between $\mathbf{q}, \mathbf{p_i}$. A typical problem, which we term as the *threshold kernel aggregation query* (τKAQ), is to test whether $\mathcal{F}_P(\mathbf{q})$ is higher than a given threshold τ [30]. This creates an opportunity for achieving speedup. Instead of computing the exact $\mathcal{F}_P(\mathbf{q})$, it suffices to compute lower/upper bounds of $\mathcal{F}_P(\mathbf{q})$ and then compare them with the threshold τ.

In addition, different types of weighting (for w_i) have been used in different statistical/learning models, as summarized in Table I. Although there exist several techniques to speedup the computation of $\mathcal{F}_P(\mathbf{q})$, each work focuses on one type of weighting only [14], [13], [18], [16], [21]. In contrast, this paper intends to handle the kernel aggregation query under all types of weightings.

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

Type of weighting	Used in model	Techniques
Type I: identical, positive w_i (most specific)	Kernel density [14], [13]	Quality-preserving solutions [14], [13]
Type II: positive w_i (subsuming Type I)	1-class SVM [26]	Heuristics [23]
Type III: no restriction on w_i (subsuming Types I, II)	2-class SVM [30]	Heuristics [18], [16], [21]

The above query is 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 works [21], [16], [18] also complain the inefficiency issue 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."* [21]
- *"However, computing the decision function for the new test samples is typically expensive which limits the applicability of kernel methods to real-world applications."* [16]
- *"..., it has the disadvantage of requiring relatively large computations in the testing phase"* [18]

Existing solutions are divided into two camps. The machine learning community tends to improve the response time by using heuristics [18], [16], [21], [23] (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 [14], [13], albeit they are only applicable to queries with Type I weighting (see Table I). Their idea [14], [13] 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.

In this paper, we identify several important research issues that have not yet been addressed in [14], [13], as listed below:

1) **Tighter bound functions:** How to design lower/upper bound functions that are always tighter than existing ones? How to compute them quickly?
2) **Type of weighting:** The techniques in [14], [13] are applicable to Type I weighting only (see Table I). Can we develop a general solution for all types of weighting?
3) **Automatic index tuning:** The performance of a solution may vary greatly across different types of index structures. How to develop an automatic index tuning technique for achieving the best possible efficiency?
4) **In-situ scenario:** In this scenario, the entire dataset is not known in advance. An example scenario is online kernel learning [11], [24], [20], in which the model (e.g., dataset P) would be updated frequently. The end-to-end response time includes the index construction time and the tuning time as well. How to develop a quick tuning technique while enjoying the benefit of a reasonably-good index structure?

Our contributions are twofold. Our proposal is **K**ernel **A**ggregation **R**apid **L**ibrary (KARL)[1], a comprehensive solution for addressing all the issues mentioned above. Experimental studies on many real datasets reveal that our proposed method achieves speedups of 2.5-800 over the state-of-the-art.

Two widely-used libraries, namely LibSVM [7] and Scikit-learn [28], provide convenient programming support for practitioners to handle kernel aggregation queries. Implementation-wise, LibSVM is based on the sequential scan method, and Scikit-learn is based on the algorithm in [14] in query type I. We compare them with our proposal (KARL) in Table II. As a remark, Scikit-learn also supports query types II and III. However, the algorithm is just the wrapper of LibSVM algorithm [28]. As such, we do not compare Scikit-learn in query types II and III. The features of KARL are: (i) it supports all three types of weightings as well as both ϵKAQ and τKAQ queries, (ii) it supports index structures, (iii) it yields much lower response time than existing libraries.

We first introduce the preliminaries in Section II, and then present our solution in Section III. We later extend our techniques to different types of weighting and kernel functions in Section IV. After that, we present our experiments in Section V. Then, we present our related work in Section VI. Lastly, we conclude the paper with future research directions in Section VII.

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

TABLE II: Comparisons of libraries

Library	Supported queries	Supported weightings	Support indexing	Response time
LibSVM [7]	τKAQ	Types I, II, III	no	medium
Scikit-learn [28]	ϵKAQ	Type I	yes	high
KARL (this paper)	ϵKAQ, τKAQ	Types I, II, III	yes	low

II. Preliminaries

We consider two popular types of kernel aggregation queries in the literature [14], [30]. The first variant is to compute an approximate value of $\mathcal{F}_P(\mathbf{q})$ with accuracy bound [14]. We call this as *approximate kernel aggregation query* (ϵKAQ), which returns an approximate value within $(1 \pm \epsilon)$ times the exact value of $\mathcal{F}_P(\mathbf{q})$. The second variant is to simply test whether $\mathcal{F}_P(\mathbf{q})$ is higher than a threshold [30]. We term this as the *threshold kernel aggregation query* (τKAQ), which simply tests whether $\mathcal{F}_P(\mathbf{q}) \geq \tau$, where τ is a given threshold.

A. Problem Statement

First, we reiterate the kernel aggregation query (KAQ) as discussed in the introduction.

Definition 1 (KAQ). *Given a query point* $\mathbf{q}$ *and a set of points* P*, this query computes:*

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \, \mathcal{K}(\mathbf{q}, \mathbf{p_i}) \tag{2}$$

where w_i *is a scalar indicating the weight of the* i*-th term, and* $\mathcal{K}(\mathbf{q}, \mathbf{p_i})$ *denotes the kernel function.*

In the machine learning and statistics communities [7], [30], [33], the typical kernel functions are the Gaussian kernel function, the polynomial kernel function, and the sigmoid kernel function. For example, the Gaussian kernel function is expressed as $\mathcal{K}(\mathbf{q}, \mathbf{p_i}) = \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2)$, where γ is a positive scalar denoting smoothing parameter, and $dist(\mathbf{q}, \mathbf{p_i})$ denotes the Euclidean distance between $\mathbf{q}, \mathbf{p_i}$.

Then we formally define two variants of KAQ: *threshold kernel aggregation query* (τKAQ) [30] and *approximate kernel aggregation query* (ϵKAQ) [14].

Problem 1 (τKAQ). *Given a threshold value* τ*, a query point* $\mathbf{q}$*, and a set of points* P*, this problem returns a Boolean value denoting whether* $\mathcal{F}_P(\mathbf{q}) \geq \tau$.

Problem 2 (ϵKAQ). *Given a relative error value* ϵ*, a query point* $\mathbf{q}$*, and a set of points* P*, this problem returns an approximate value* $\widehat{F}$ *such that its relative error (from the exact value* $\mathcal{F}_P(\mathbf{q})$*) is at most* ϵ*, i.e.,*

$$(1-\epsilon)\mathcal{F}_P(\mathbf{q}) \leq \widehat{F} \leq (1+\epsilon)\mathcal{F}_P(\mathbf{q}) \tag{3}$$

Table III summarizes the types of queries that can be used for each application model. Table IV summarizes the frequently-used symbols in this paper.

TABLE III: Example applications for the above queries

Application model	Relevant queries	Obtained from training/learning	Specified by user
Kernel density [14], [13]	ϵKAQ, τKAQ	N.A.	query point $\mathbf{q}$, point set P, parameters ϵ, τ, γ
1-class SVM [26]	τKAQ	point set P, weights w_i, parameters τ, γ	query point $\mathbf{q}$
2-class SVM [30]	τKAQ	point set P, weights w_i, parameters τ, γ	query point $\mathbf{q}$

TABLE IV: Symbols

Symbol	Description
P	Point set
$\mathcal{F}_P(\mathbf{q})$	Kernel aggregation function (Equation 2)
$\mathcal{K}(\mathbf{q}, \mathbf{p})$	Kernel (e.g. Gaussian, polynomial)
$Lin_{m,c}(x)$	Linear function $mx + c$
$\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$	Linear lower bound of $\mathcal{F}_P(\mathbf{q})$
$\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u})$	Linear upper bound of $\mathcal{F}_P(\mathbf{q})$
$dist(\mathbf{q}, \mathbf{p})$	Euclidean distance between $\mathbf{q}$ and $\mathbf{p}$

B. State-of-the-Art (SOTA)

We proceed to introduce the state-of-the-art [14], [13] (SOTA), albeit they are only applicable to queries with Type I weighting (see Table I). In this case, we denote the common weight by w.

Bounding functions.

We introduce the concept of *bounding rectangle* [29] below.

Definition 2. *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.

It holds that $mindist(\mathbf{q}, R) \leq dist(\mathbf{q}, \mathbf{p}) \leq maxdist(\mathbf{q}, R)$ for every point $\mathbf{p}$ inside 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 1) are defined as:

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

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

Refinement of bounds.

The state-of-the-art [14], [13] 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).

We illustrate the running steps of the state-of-the-art on the above example index in Table V. For conciseness, the notations

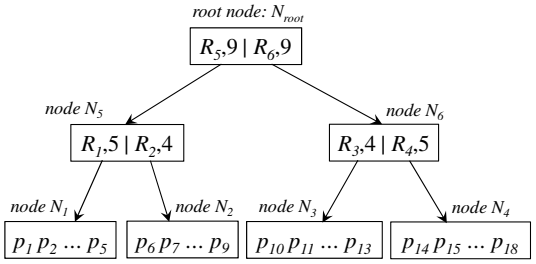


Fig. 2: Hierarchical index structure

$LB_R(\mathbf{q}), UB_R(\mathbf{q}), \mathcal{F}_P(\mathbf{q})$ 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, process the child entries of R_i, then refine the bounds incrementally and update 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 refine the bounds incrementally.

TABLE V: 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 ϵKAQ, the termination condition is: $\widehat{ub} < (1+\epsilon)\widehat{lb}$. For τKAQ, the termination condition is: $\widehat{lb} \geq \tau$ or $\widehat{ub} < \tau$.

III. Our Solution: KARL

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

Our key contribution is to develop tighter bound functions for $\mathcal{F}_P(\mathbf{q})$. In Section III-A, we propose a novel idea to bound the function $\exp(-x)$ and discuss how to compute such bound functions quickly. In Section III-B, we devise tighter bound functions and show that they are tighter than existing bound functions. Then, we discuss automatic tuning in Section III-C.

In this section, we assume using Type I weighting and the Gaussian kernel in the function $\mathcal{F}_P(\mathbf{q})$. We leave the extensions to other types of weighting and kernel functions in Section IV.

A. Fast Linear Bound Functions

We wish to design bound functions such that (i) they are tighter than existing bound functions (cf. Section II-B), and (ii) they are efficient to compute, e.g., 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 denotes the value $\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2$. With this notation, the value $\mathcal{F}_P(\mathbf{q})$ 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.

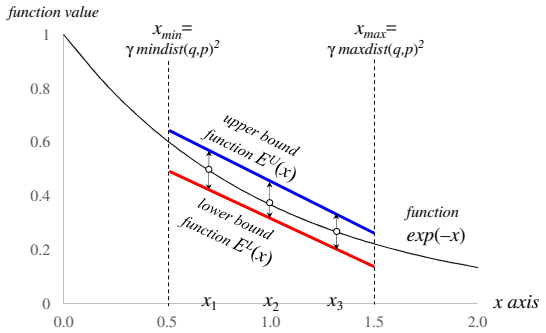


Fig. 3: Linear bounds

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 (see Definition 3). We illustrate these two functions by a red line and a blue line in Figure 3.

Definition 3 (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 $Lin_{m,c}$ 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 $\mathcal{F}_P(\mathbf{q})$, subject to the condition stated in the following lemma:

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

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

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, i.e., 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. □

The equation in Lemma 2 provides an efficient way to compute $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$.

B. Tighter Bound Functions

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

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

Recall that, by using the query point $\mathbf{q}$ and the bounding rectangle R (of point set P), we obtain the bounding interval $[x_{min}, x_{max}]$, where $x_{min} = \gamma \cdot mindist(\mathbf{q}, R)^2$ and $x_{max} = \gamma \cdot maxdist(\mathbf{q}, R)^2$. Since $\exp(-x)$ is a convex function, the chord between two points (say, $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$ must always reside above the curve $\exp(-x)$. We illustrate this in Figure 4.

Regarding the linear function Lin_{m_u,c_u}, its slope m_u and the intercept c_u are computed as:

$$m_u = \frac{\exp(-x_{max}) - \exp(-x_{min})}{x_{max} - x_{min}} \quad (6)$$

$$c_u = \frac{x_{max}\exp(-x_{min}) - x_{min}\exp(-x_{max})}{x_{max} - x_{min}} \quad (7)$$

It turns out that the above chord-based linear function Lin_{m_u,c_u} leads to a tighter upper bound than the existing

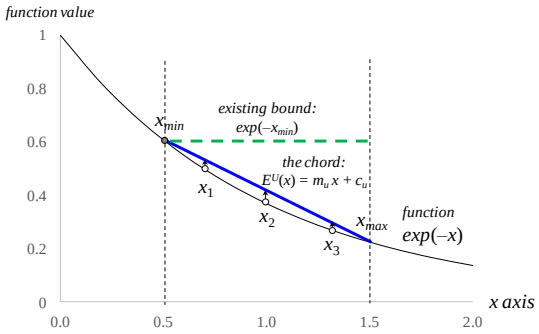


Fig. 4: Chord-based upper bound function

bound $\exp(-x_{min})$ (see Section II-B). Clearly, as shown in Figure 4, the projected values on the blue line (Lin_{m_u,c_u}) are smaller than the existing bound $\exp(-x_{min})$ (green dashed line in Figure 4).

Lemma 3. *There exists a linear function Lin_{m_u,c_u} such that $\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u}) \leq UB_R(\mathbf{q})$, where $UB_R(\mathbf{q})$ is the upper bound function used in the state-of-the-art (see Section II-B).*

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

We exploit a property of convex function [15], namely that, any tangent line of a convex function must be a lower bound of the function. This property is applicable to $\exp(-x)$ because it is also a convex function.

We illustrate the above property in Figure 5a. For example, the tangent line of function $\exp(-x)$ at point $(x_{max}, \exp(-x_{max}))$ serves as the lower bound function for $\exp(-x)$. Furthermore, this lower bound is already tighter than the existing bound $\exp(-x_{max})$ (see Section II-B). Note that in Figure 5a, the projected values on the red line (Lin_{m_l,c_l}) are higher than the existing bound $\exp(-x_{max})$ (green dashed line in Figure 5a).

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

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 lower bound linear function Lin_{m_l,c_l} is the tangent line at point $(t, \exp(-t))$. Then, we derive the slope m_l and the intercept c_l as:

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

The following theorem 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:*

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

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:

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

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.

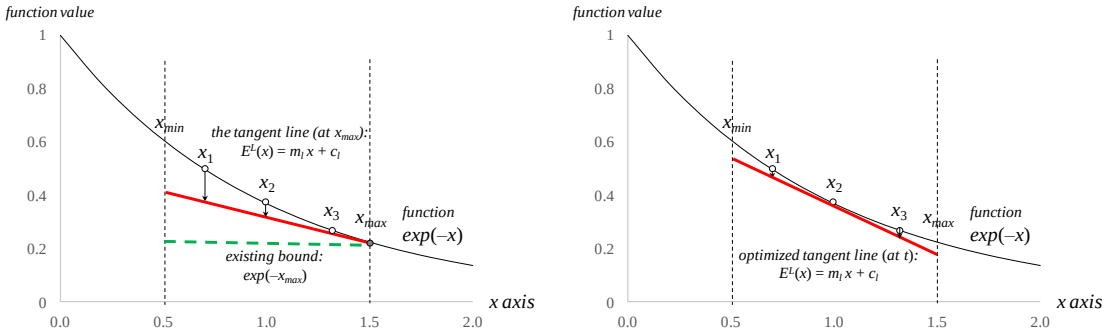
1) For the case $t > t_{opt}$, we get $H'(t) < 0$, implying that $H(t)$ is a decreasing function
2) For the case $t < t_{opt}$, we get $H'(t) > 0$, implying that $H(t)$ is an increasing function.

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 V-A 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.



(a) tangent line at x_{max} (b) optimized tangent line at t

Fig. 5: Tangent-based lower bound function

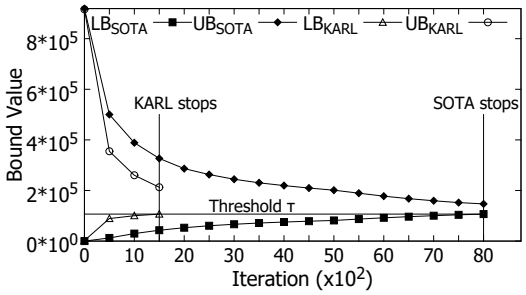
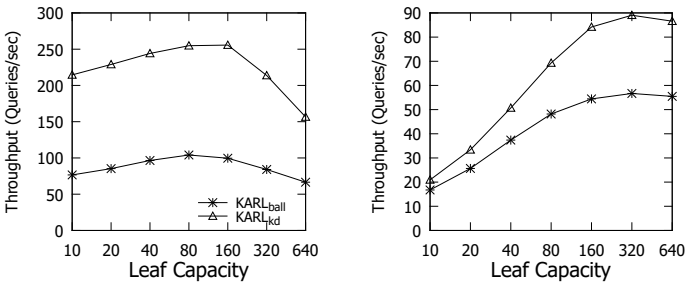


Fig. 6: Bound values of SOTA and KARL vs. the number of iterations; for type I-τ query on the home dataset



(a) dataset home (b) dataset susy

Fig. 7: The throughput of query type I-τ, varying the leaf node capacity

C. Automatic Tuning

The performance of KARL depends on the choices of the index structure and the index height. Popular index structures include the k-d tree and the ball tree, which are also supported in an existing machine learning library (e.g., Scikit-learn [28]). In addition, the height of such index structure can be controlled via the parameter 'leaf node capacity' (i.e., the maximum number of points per leaf node).

To demonstrate the above effect, we conduct the following test by using different index structures with different values of leaf node capacity. Then we measure the throughput (i.e., the number of processed queries per second) of KARL in each index structure. Figures 7a,b show the throughput of KARL on two datasets (home and susy) respectively. In each figure, the speedup of the best choice to the worst choice can be up to 4 times. Furthermore, the optimal choice can be different on different datasets.

To tackle the above issue, we propose some automatic tuning techniques.

Offline tuning scenario.

In this scenario, we are given ample time for tuning and the dataset is provided in advance.

Observe that two index structures with similar leaf node capacity (e.g., 100 and 101) tend to offer similar performance. It is sufficient to vary the leaf node capacity in an exponential manner (e.g., 10,20,40,80,160,320,640). Next, we build an index structure for each parameter value and for each index type. Finally, we sample a small subset Q of query points, then recommend the index structure having the highest throughput on Q. According to our experimental results, it is enough to set the sample size to $|Q| = 1000$.

In-situ scenario: online tuning.

This scenario is more challenging because the dataset is not known in advance. The end-to-end response time includes index construction time, tuning time, and query execution time. To achieve high throughput, we should reduce the index construction time and the tuning time, while figuring out a reasonably good index.

First, we recommend to build the k-d tree due to its low construction time. It suffices to build a single k-d tree with all levels (i.e., $\log_2(n)$ levels). Denote the entire tree by $\mathcal{T}$, and the tree with the top i levels by $\mathcal{T}_i$. Observe that the tree $\mathcal{T}_i$ can be simulated by using the entire tree $\mathcal{T}$, by skipping lower/upper bound computations in the lowest $\log_2(n) - i$ levels of $\mathcal{T}$.

Suppose that we are given the number of queries to be executed. We sample $s\%$ (say, 1%) of those queries and then partition them into $\log_2(n)$ groups. For the i-group, we run its sample queries on the tree $\mathcal{T}_i$. Then, we obtain the value i^* that yields the best performance. Finally, we execute the remaining (100-s)% of queries on the k-d tree $\mathcal{T}_{i^*}$.

IV. Extensions

The state-of-the-art solution has not considered other types of weighting nor other types of kernel functions. In this section, we adapt our bounding techniques (in Sections III-A and III-B) to address these issues.

A. Other Types of Weighting

We extend our bounding techniques for the following function:

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

under other types of weighting.

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 $Lin_{m,c}$ 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 the following lemma.

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 precomputed 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.

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:*

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

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.

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

Then, we follow exactly the same steps of Theorem 1 to derive the maximum value (Equation 10). □

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

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:

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

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

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

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 $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$.

B. Other Kernel Functions

In this section, we develop our bounding techniques for other kernel functions, such as the polynomial kernel function, and the sigmoid kernel function.

First, we consider the the polynomial kernel function $\mathcal{K}(\mathbf{q}, \mathbf{p_i}) = (\gamma\ \mathbf{q} \cdot \mathbf{p_i} + \beta)^{deg}$, where γ, β are scalar values, and deg denotes the polynomial degree. In this context, we express the function $\mathcal{F}_P(\mathbf{q})$ as follows.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i (\gamma\ \mathbf{q} \cdot \mathbf{p_i} + \beta)^{deg} \quad (11)$$

For the sake of discussion, we assume Type II weighting (i.e., positive weight coefficients w_i).

We introduce the notation x_i to represent the term $\gamma\ \mathbf{q} \cdot \mathbf{p_i} + \beta$. The bounding interval of x_i, i.e., the interval $[x_{min}, x_{max}]$, is computed as:

$$\begin{aligned} x_{min} &= \gamma\ \mathbb{IP}_{min}(\mathbf{q}, R) + \beta \\ x_{max} &= \gamma\ \mathbb{IP}_{max}(\mathbf{q}, R) + \beta \end{aligned}$$

where R is the bounding rectangle of P, and $\mathbb{IP}_{min}(\mathbf{q}, R), \mathbb{IP}_{max}(\mathbf{q}, R)$ represent the minimum and the maximum inner product between $\mathbf{q}$ and R, respectively.

We then extend the efficient computation techniques in Section III-A. Suppose that we are given 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 $Lin_{m_l,c_l}(x) \leq x^{deg} \leq Lin_{m_u,c_u}(x)$ for all x. Similar to Lemma 1, we can obtain the following property: $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l}) \leq \mathcal{F}_P(\mathbf{q}) \leq \mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u})$, where:

$$\begin{aligned}\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) &= \sum_{\mathbf{p_i} \in P} w_i(m(\gamma \mathbf{q} \cdot \mathbf{p_i} + \beta) + c) \\ &= m\gamma \mathbf{q} \cdot \hat{\mathbf{a_P}} + (m\beta + c)\hat{b_P}\end{aligned}$$

where $\hat{\mathbf{a_P}} = \sum_{\mathbf{p_i} \in P} w_i \mathbf{p_i}$ and $\hat{b_P} = \sum_{\mathbf{p_i} \in P} w_i$. Such function can be computed in $O(d)$ time, provided that the terms $\hat{\mathbf{a_P}}, \hat{b_P}$ have been precomputed.

We proceed to discuss how to devise the bounding linear functions for x^{deg}. When deg is even, the function x^{deg} satisfies the convex property, and thus the techniques in Section III-B remain applicable. However, when deg is odd, the techniques in Section III-B are not applicable. For example, we plot the function x^3 in Figure 8 and notice that the chord between x_{min} and x_{max} is no longer an upper bound function.

Our idea is to exploit the monotonic increasing property of the function x^{deg} (given that deg is odd). We illustrate how to construct the bounding linear functions in Figure 8. For the upper bound function, we start with the horizontal line $y = (x_{max})^{deg}$ and then rotate-down the line until its left-hand-side hits the function x^{deg}. For the lower bound function, we start with the horizontal line $y = (x_{min})^{deg}$ and then rotate-up the line until its right-hand-side hits the function x^{deg}. The parameters of these lines can be derived by mathematical techniques.

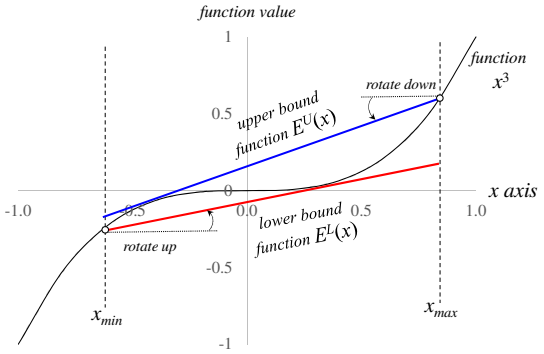


Fig. 8: Lower and upper bound functions for x^3

The above idea is also applicable to the sigmoid kernel because it is also a monotonic increasing function. The tightness of all these bound functions depends on x_{min} and x_{max}.

V. Experimental Evaluation

We introduce the experimental setting in Section V-A. Next, we demonstrate the performance of different methods in Section V-B. Then, we compare the tightness of KARL and SOTA bound functions in Section V-C. After that, we perform experimental analysis of our index-tuning method (KARL$_{\text{auto}}$) in Section V-D. Next, we compare different methods for in-situ applications in Section V-E. Lastly, we extend our techniques to combine with polynomial kernel function in Section V-F.

A. Experimental Setting

1) Datasets: For Type-I, Type-II, Type-III weighting, we take the application model as kernel density estimation, 1-class SVM, and 2-class SVM, respectively. We use a wide variety of real datasets for these models, as shown in Table VI. The value n_{raw} denotes the number of points in the raw dataset, and the value d denotes the data dimensionality. The source/reference for each dataset is also provided. These datasets either come from data repository websites [2], [7] or have been used in recent papers [13], [12].

For Type-I weighting, we follow [13] and use the Scott's rule to obtain the parameter γ. Type-II and Type III datasets require a training phase. We consider two kernel functions: the Gaussian kernel and the polynomial kernel. We denote the number of remaining points in the dataset after training as n_{model}^{gauss} and n_{model}^{poly}, for the Gaussian kernel and the polynomial kernel respectively.

The LIBSVM software [7] 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 $\gamma = \frac{1}{d}$ [7]. Then we vary the training model parameter ν 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 training with the automatic script in [7] to determine the suitable values for training parameters.

TABLE VI: Details of datasets

Model	Raw dataset	n_{raw}	n_{model}^{gauss}	n_{model}^{poly}	d
Type I: kernel density	mnist [7]	60000	n/a	n/a	784
	miniboone [2]	119596	n/a	n/a	50
	home [2]	918991	n/a	n/a	10
	susy [2]	4990000	n/a	n/a	18
Type II: 1-class SVM	nsl-kdd [1]	67343	17510	6738	41
	kdd99 [2]	972780	19461	19462	41
	covtype [7]	581012	25486	14165	54
Type III: 2-class SVM	ijcnn1 [7]	49990	9592	9706	22
	a9a [7]	32561	11772	15682	123
	covtype-b [7]	581012	310184	323523	54

2) Methods for Comparisons: SCAN is the sequential scan method which computes $\mathcal{F}_P(\mathbf{q})$ without any pruning. Scikit-learn (abbrev. Scikit) is the machine learning library which supports the approximate KDE problem [14] (i.e. query type I-ϵ) and handles SVM-based classification (by LIBSVM [7]), i.e. query type I-τ, types II-τ and III-τ. SOTA is the state-of-the-art method which was developed by [13] for handling the Kernel Density Classification problem, i.e. I-τ query. We modify and extend their framework to handle other types of queries. Our KARL follows the framework of [13], combining with our linear bound functions, LB_{KARL} and UB_{KARL}. Both SOTA and KARL can work seamlessly with various index-structures. The space complexity of all these methods

are $O(|P|d)$. Even for the largest tested dataset (susy), the memory consumption is only at most 1.34GBytes.

For the indexing options, kd-tree [29] and ball-tree [32], [27] are currently supported by Scikit. We only report the best result of Scikit (denoted as $\text{Scikit}_{\text{best}}$). For consistency, we also evaluate SOTA and KARL with these two indices. KARL can automatically choose suitable index and leaf capacity among these two indices, which we called $\text{KARL}_{\text{auto}}$. To demonstrate our effectiveness compared with SOTA, we select the best index with the best leaf capacity during the comparison in later sections, which we denote it by $\text{SOTA}_{\text{best}}$. For in-situ application, we combine the online-tuning method with KARL which we called $\text{KARL}_{\text{auto}}^{\text{online}}$. We also compare this method with the best performance of SOTA for this scenario, which we term it as $\text{SOTA}_{\text{best}}^{\text{online}}$.

We implemented all algorithms in C++ and conducted experiments on an Intel i7 3.4GHz PC running Ubuntu. For each dataset, we randomly sample 10,000 points from the dataset as the query set Q. Following [13], we measure the efficiency of a method as the throughput (i.e., the number of queries processed per second).

B. Efficiency Evaluation for Different Query Types

We test the performance of different methods in four types of queries which are I-ϵ, I-τ, II-τ and III-τ. The parameters of these queries are set as follows.

Type I-ϵ. We set the relative error $\epsilon = 0.2$ for each dataset.

Type I-τ. We fix the mean value μ 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 τ for each dataset in Table VII.

Types II-τ and III-τ. The threshold τ can be obtained during the training phase.

TABLE VII: All methods for different types of queries

Type	Datasets	SCAN	LIBSVM	$\text{Scikit}_{\text{best}}$	$\text{SOTA}_{\text{best}}$	$\text{KARL}_{\text{auto}}$
I-ϵ	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-τ	miniboone	36.1	34	n/a	102	**510**
	home	15.2	14.1	n/a	93.2	**258**
	susy	2.02	1.86	n/a	3.58	**83.4**
II-τ	nsl-kdd	283	481	n/a	748	**20668**
	kdd99	260	520	n/a	1269	**11324**
	covtype	158	462	n/a	448	**6022**
III-τ	ijcnn1	903	1170	n/a	1119	**826928**
	a9a	162	610	n/a	546	**6885**
	covtype-b	13	38.4	n/a	33.9	**274**

Table VII shows the throughput of different methods for all types of queries. In the result of query type I-ϵ, SCAN is comparable to $\text{Scikit}_{\text{best}}$ and $\text{SOTA}_{\text{best}}$ since the bounds computed by the basic bound functions are not tight enough. The performance of $\text{Scikit}_{\text{best}}$ and $\text{SOTA}_{\text{best}}$ is affected by the overhead of the loose bound computations. $\text{KARL}_{\text{auto}}$ uses our new bound functions which are shown to provide tighter bounds. These bounds lead to significant speedup in all evaluated datasets, e.g., $\text{KARL}_{\text{auto}}$ is at least 5.16 times faster than other methods.

For the type I-τ threshold-based queries, our method $\text{KARL}_{\text{auto}}$ improves the throughput by 2.76x to 21.2x when comparing to the runner-up method SOTA. The improvement becomes more obvious in type II-τ and type III-τ queries. The improvement of $\text{KARL}_{\text{auto}}$ can be up to 738x as compared to SOTA. This is because the number of remaining points (cf. Table VI) becomes sparse as compared to the raw datasets. The query search space can be dramatically reduced on such sparse datasets if tighter bounds are provided. Thereby, $\text{KARL}_{\text{auto}}$ is superior to other methods in these two types of queries.

Sensitivity of τ. In order to test the sensitivity of threshold τ in different methods, we select seven thresholds from the range $\mu - 2\sigma$ to $\mu + 4\sigma$, where $\sigma = \sqrt{\sum_{\mathbf{q}\in Q}(\mathcal{F}_P(\mathbf{q}) - \mu)^2/|Q|}$ is the standard deviation. Figure 9 shows the results on three datasets. As a remark, for the miniboone dataset, we skip the thresholds $\mu - \sigma$ and $\mu - 2\sigma$ as they are negative. Due to the superior performance of our bound functions, $\text{KARL}_{\text{auto}}$ outperforms $\text{SOTA}_{\text{best}}$ by nearly one order of magnitude in most of the datasets regardless of the chosen threshold.

Sensitivity of ϵ. In Scikit-learn library [28], we can select different relative error ϵ, which is called as tolerance in the approximate KDE. To test the sensitivity, we vary the relative error ϵ for different datasets with query type I-ϵ. If the value of ϵ is very small, the room for the bound estimations is very limited so that neither $\text{KARL}_{\text{auto}}$ nor $\text{SOTA}_{\text{best}}$ perform well in very small ϵ setting (e.g., 0.05). For other general ϵ settings, our method $\text{KARL}_{\text{auto}}$ consistently outperforms other methods by a visible margin (c.f. Figure 10).

Sensitivity of dataset size. In the following experiment, we test how the size of the dataset affects the evaluation performance of different methods in both query types I-ϵ and I-τ. We choose the largest dataset (susy) and vary the size via sampling. The trend in Figure 11 meets our expectation; a smaller size implies a higher throughput. Our $\text{KARL}_{\text{auto}}$ in general outperforms other methods by one order of magnitude in a wide range of chosen size of datasets.

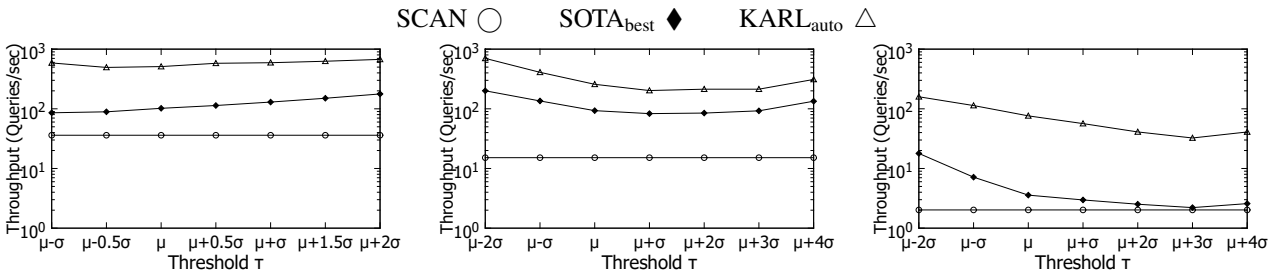
Sensitivity of dimensionality. In this experiment, we choose the dataset (mnist) with the largest dimensionality (784) and then vary the dimensionality via PCA dimensionality reduction as in [13]. The default threshold $\tau = \mu$ is used. As shown in Figure 12, our method $\text{KARL}_{\text{auto}}$ consistently outperforms existing methods under different dimensionalities.

C. Tightness of Bound Functions

Recall from Section III, we have theoretically shown that our developed linear bound functions LB_{KARL} and UB_{KARL} are tighter than SOTA bound functions. In this section, we explore how tight our bound functions can be better than the SOTA in practice.

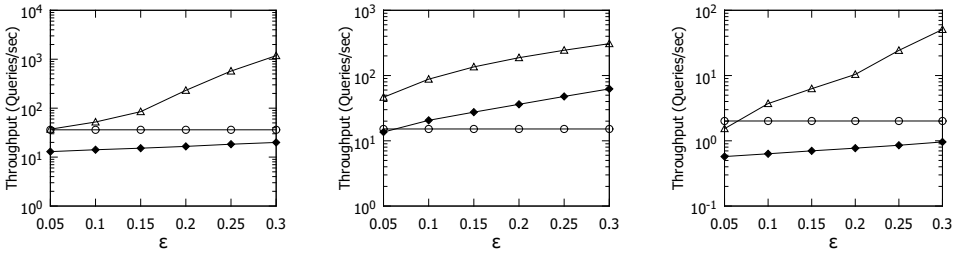
For the sake of fairness, we fix the tree-structure to be the kd-tree with leaf capacity 80. We use the following equation to measure the average tightness of bound values.

$$Error_{bound} = \frac{1}{|Q| \cdot L} \sum_{l=1}^{L} \sum_{\mathbf{q}\in Q} \left| \frac{\sum_{G_j \in \mathbf{G}_l} bound(\mathbf{q}, G_j) - \mathcal{F}_P(\mathbf{q})}{\mathcal{F}_P(\mathbf{q})} \right|$$



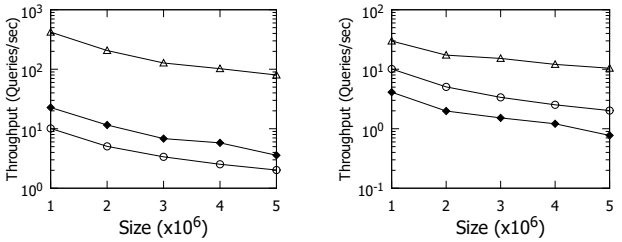
(a) miniboone (b) home (c) susy

Fig. 9: Query throughput with query type I-τ, varying the threshold τ



(a) miniboone (b) home (c) susy

Fig. 10: Query throughput with query type I-ϵ, varying the relative error ϵ



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

Fig. 11: Query throughput on the susy dataset, varying the dataset size

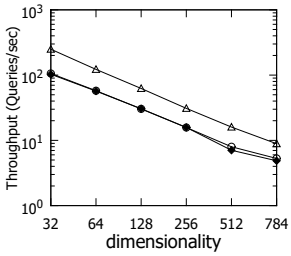
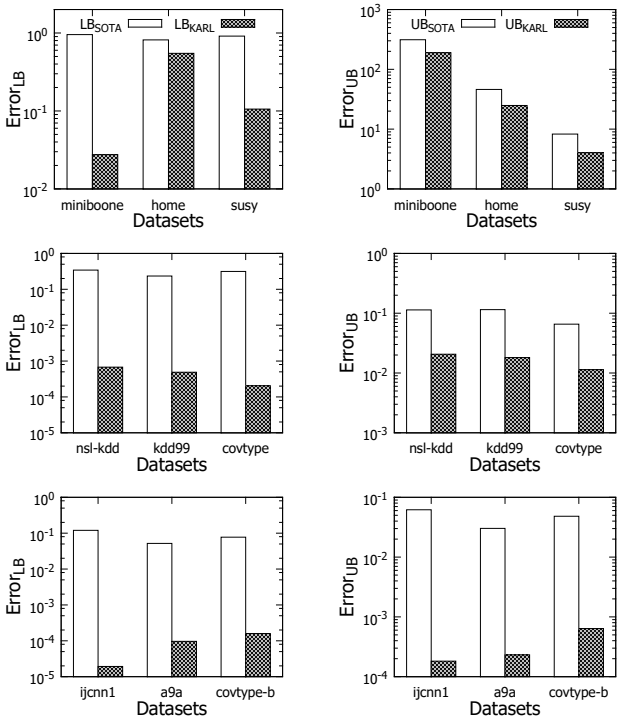


Fig. 12: Query throughput with query type I-τ ($\tau = \mu$) on the mnist dataset, varying the dimensionality



Type-I (1st row); Type-II (2nd row); Type-III (3rd row)

Fig. 13: $Error_{LB}$ and $Error_{UB}$ in Type-I, II and III query

Our bound functions are in practice much tighter than the LB_{SOTA} and UB_{SOTA} in all evaluated datasets, especially for LB_{KARL}, as shown in Figure 13. In addition, the bound functions (of SOTA and KARL) provide very tight bounds for query types II and III. There are two reasons for this phenomenon, including the data distribution and data normalization. First, the data point of types II and III are the support vectors (being trained by SVM), which are the nearest points to the decision boundary [30] and are very close to each other.

Second, the set of support vectors are normalized from 0 to 1 [7]. This limits the range of possible values of the exponential function so that the lower and upper bounds are close to the exact value.

D. Index auto-tuning (Offline)

Recall from Figure 2, different indexing structures can be applied to our problems. One natural question is how can we predict the suitable index-structure. In this section, we demonstrate the method $\text{KARL}_{\text{auto}}$, which automatically selects the best tree-structure with the suitable leaf capacity from kd-tree [29] and ball-tree [32], [27]. These two tree-structures are currently supported by Scikit-learn library [28].

Our solution $\text{KARL}_{\text{auto}}$ randomly samples 1000 vectors, denoted by the sample set S, from each dataset and predicts the performance based on the throughput, using different leaf capacities of tree. Table VIII shows that in the offline stage, our method $\text{KARL}_{\text{auto}}$ can provide good prediction which can make the online throughput near the best solution $\text{KARL}_{\text{best}}$.

TABLE VIII: Query throughput for variants of KARL, using sample set with $|S| = 1000$

Type	Datasets	$\text{KARL}_{\text{worst}}$	$\text{KARL}_{\text{auto}}$	$\text{KARL}_{\text{best}}$
I-ϵ	miniboone	88.1	302	304
	home	35.9	185	188
	susy	5.5	12.9	13.3
I-τ	miniboone	64.8	514	566
	home	76.6	258	258
	susy	16.7	84	89
II-τ	nsl-kdd	4357	20668	20677
	kdd99	5911	11324	11325
	covtype	915	6022	6038
III-τ	ijcnn1	388109	826928	843601
	a9a	408	6885	6891
	covtype-b	52	274	277

E. Solution for In-situ applications

In some online learning scenarios [11], [24], [20], we may not be able to pre-build the index. Thereby, we cannot simply omit the index construction time. In this section, we consider the online-tuning solution. The algorithm first builds the k-d tree with all levels ($\log_2 n$). Then it samples 1% of queries, which are further partitioned into $\log_2 n$ query groups. For each group i, we select tree level i as the leaf level and evaluate the throughput. Lastly, we choose tree-level i^* as the leaf level for remaining queries in which the query group i^* yields the highest throughput. All results are shown in Table IX.

For query types I-ϵ and I-τ, $\text{SOTA}_{\text{best}}^{\text{online}}$ is not efficient because its bounding functions are not tight (recall from Figure 6). Our method $\text{KARL}_{\text{auto}}^{\text{online}}$ outperforms existing methods significantly on all tested datasets.

For query types II-τ and III-τ, our $\text{KARL}_{\text{auto}}^{\text{online}}$ can significantly increase the throughput in several datasets since each support vector in the datasets is near with each other.

F. Polynomial Kernel

Recall from previous section, our linear bound functions can be also used for polynomial kernel. In this section, we experimentally test the online query throughput with Type-II query. We use the polynomial kernel with degree 3 which is the default setting in LIBSVM [7]. We also normalize the datasets in the range [-1,1] [7]. Then, we apply the same setting in the training phases in 1-class and 2-class SVM which are stated in Section V-B. Table X shows that our method $\text{KARL}_{\text{auto}}$ outperforms $\text{SOTA}_{\text{best}}$ by 3x to 11x times.

TABLE IX: In-situ solutions for different types of queries

Type	Datasets	baseline	$\text{SOTA}_{\text{best}}^{\text{online}}$	$\text{KARL}_{\text{auto}}^{\text{online}}$
I-ϵ	miniboone	36.1	16.4	**217**
	home	35.8	32.1	**184**
	susy	2.02	0.75	**7.26**
I-τ	miniboone	36.1	101	**419**
	home	15.2	92.1	**243**
	susy	2.02	3.57	**51**
II-τ	nsl-kdd	481	733	**9869**
	kdd99	260	1264	**7920**
	covtype	462	439	**2389**
III-τ	ijcnn1	1170	1112	**426132**
	a9a	610	543	**1966**
	covtype-b	38.4	33.5	**101**

TABLE X: Query throughput with query type II/III-τ using polynomial kernel

Type	Datasets	baseline	$\text{SOTA}_{\text{best}}$	$\text{KARL}_{\text{auto}}$
II-τ	nsl-kdd	909	1200	**4522**
	kdd99	314	639	**2741**
	covtype	537	6423	**88396**
III-τ	ijcnn1	1154	1122	**185372**
	a9a	463	422	**2813**
	covtype-b	36.4	30.5	**187**

VI. Related Work

The term "kernel aggregation query" abstracts a common operation in several statistical and learning problems such as kernel density estimation [14], [13], 1-class SVM [26], and 2-class SVM [30].

Kernel density estimation is a non-parametric statistical method for density estimation. To speedup kernel density estimation, existing works would either compute approximate density values with accuracy guarantee [27] or test whether density values are above a given threshold [13]. Zheng et al. [35] focus on fast kernel density estimation on low-dimensional data (e.g., 1d, 2d) and propose sampling-based solutions with theoretical guarantees on both efficiency and quality. On the other hand, [27], [13] 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 [28] adopts the implementation in [27]. Our proposal, KARL, adapts the algorithm in [27], [13] to evaluate kernel aggregation queries. The key difference between KARL and [27], [13] lies in the bound functions. As explained in Section III-B, our proposed linear bound functions are tighter than existing bound functions used in [27], [13]. Furthermore, we extend our linear bound functions to deal with different types of weighting and kernel functions, which have not been considered in [27], [13].

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 [26], network fault detection [34], [5], [4], anomaly/outlier detection [6], [22], novelty detection [17], [25], [31], tumor samples classification [9], image classification [8], time series classification [19]. 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, threshold kernel aggregation queries can be used to support classification or outlier detection. Two approaches have been studied to accelerate the online phase. The library LibSVM [7] assumes sparse data format and applies inverted index to speedup exact computation. The machine learning community has proposed heuristics [18], [16], [21], [23] 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 works.

VII. Conclusions

In this paper, we study kernel aggregation queries, which can be used to support a common operation in kernel density estimation [14], [13], 1-class SVM [26], and 2-class SVM [30].

Our key contribution is the development of fast linear bound functions, which are proven to be tighter than existing bound functions, yet allowing fast computation. In addition, we propose a comprehensive solution that can support different types of kernel functions and weighting schemes. Our automatic tuning methods support identification of efficient index structure, which depends on the underlying point set P.

Experimental studies on a wide variety of datasets show that our solution yields higher throughput than the state-of-the-art by 2.5–800 times.

A promising future research direction is to consider more statistical/learning tasks based on kernel functions, e.g., kernel regression, multi-class kernel SVM.

References

[1] Nsl-kdd dataset. https://github.com/defcom17/.

[2] UCI machine learning repository. http://archive.ics.uci.edu/ml/index.php.

[3] Comparison of density estimation methods for astronomical datasets. *Astronomy and Astrophysics*, 531, 7 2011.

[4] 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.

[5] 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.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.

[7] 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.

[8] 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.

[9] H.-S. Chiu, S. Somvanshi, E. Patel, T.-W. Chen, V. P. Singh, B. Zorman, S. L. Patil, Y. Pan, S. S. Chatterjee, T. The Cancer Genome Atlas Research Network, A. K. Sood, P. H. Gunaratne, and P. Sumazin. Pan-Cancer Analysis of lncRNA Regulation Supports Their Targeting of Cancer Genes in Each Tumor Context. *Cell Reports*, 23(1):297–312, Apr. 2018.

[10] K. Cranmer. Kernel estimation in high-energy physics. 136:198–207, 2001.

[11] M. Davy, F. Desobry, A. Gretton, and C. Doncarli. An online support vector machine for abnormal events detection. *Signal Processing*, 86(8):2009–2025, 2006.

[12] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, 2018.

[13] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.

[14] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.

[15] O. Güler. *Foundations of Optimization*. Graduate Texts in Mathematics. Springer New York, 2010.

[16] C. Hsieh, S. Si, and I. S. Dhillon. Fast prediction for large-scale kernel machines. In *NIPS*, pages 3689–3697, 2014.

[17] 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.

[18] H. G. Jung and G. Kim. Support vector number reduction: Survey and experimental evaluations. *IEEE Trans. Intelligent Transportation Systems*, 15(2):463–476, 2014.

[19] 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.

[20] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In *NIPS*, pages 785–792, 2001.

[21] Q. V. Le, T. Sarlós, and A. J. Smola. Fastfood - computing hilbert space expansions in loglinear time. In *ICML*, pages 244–252, 2013.

[22] 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.

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

[24] J. Lu, S. C. H. Hoi, J. Wang, P. Zhao, and Z. Liu. Large scale online kernel learning. *Journal of Machine Learning Research*, 17:47:1–47:43, 2016.

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

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

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

[28] 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.

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

[30] 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.

[31] 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.

[32] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[33] M. Wand and M. Jones. *Kernel Smoothing*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[34] 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.

[35] 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.