Selectivity Functions of Range Queries are Learnable

Xiao Hu, Yuxi Liu, Haibo Xiu, Pankaj K. Agarwal, Debmalya Panigrahi, Sudeepa Roy, and Jun Yang
{xh102,yuxi.liu,haibo.xiu,pankaj,debmalya,sudeepa,junyang}@cs.duke.edu
Duke University, Durham, NC, USA

ABSTRACT

This paper explores the use of machine learning for estimating the selectivity of range queries in database systems. Using classic learning theory for real-valued functions based on shattering dimension, we show that the selectivity function of a range space with bounded VC-dimension is learnable. Since many popular classes of queries (e.g., orthogonal range search, inequalities involving linear combination of attributes, distance-based search, etc.) represent range spaces with finite VC-dimension, our result immediately implies that their selectivity functions are also learnable. To the best of our knowledge, this is the first attempt at formally explaining the role of machine learning techniques in selectivity estimation, and complements the growing literature in empirical studies in this direction. Supplementing these theoretical results, our experimental results demonstrate that, empirically, even a basic learning algorithm with generic models are able to produce accurate selectivity predictions across settings, matching state-of-art methods designed for specific query classes, and using training sample sizes commensurate with our theoretical results.

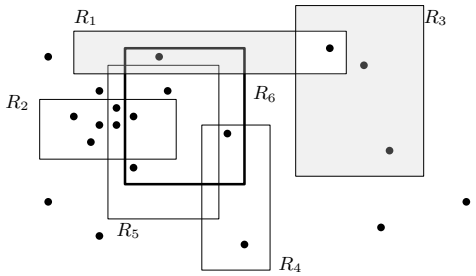

Figure 1: An illustration of the learned selectivity problem. There are 20 data points in the underlying dataset D and 5 training queries R_1, R_2, R_3, R_4, R_5 with their selectivities given by $s_D(R_1) = 0.1$, $s_D(R_2) = 0.3$, $s_D(R_3) = 0.15$, $s_D(R_4) = 0.1$ and $s_D(R_5) = 0.25$. The goal is to estimate the selectivity of an unknown query R_6 (in bold), the correct answer in this example being 0.25. The shaded area will be explained in Section 2.3.

1 INTRODUCTION

In this paper, we formally model and study the problem of learning selectivity functions for selection queries in database (DB) systems. The selectivity of a selection query on a database is defined as the probability that a randomly chosen tuple from the database satisfies the query predicate. Estimating query selectivity is a core problem in the query optimization pipeline, and has a rich history of research over many decades (see, e.g., [23, 29, 37, 38, 40]). In recent years, the focus has shifted from traditional optimization methods to machine learning (ML) techniques (e.g., [15, 24–26, 33, 36]), with the latter outperforming the former in empirical studies. In this paper, we establish a learning-theoretic framework for the selectivity-estimation problem, show that the estimation problem is indeed *learnable* for popular classes of selection queries from a small set of training samples using this framework. Building on this framework, we also develop a simple, generic learning algorithm and evaluate it empirically: not only is this approach competitive against the state-of-the-art methods designed for specific types of queries, but it also works effectively for other less-studied query types, demonstrating the power and generality of our framework.

While the query selectivity estimation problem is indeed an important component of DB research, we believe that our work also has implications beyond this specific problem. Our research adds to the growing and impressive body of work that seeks to exploit the vast advances in ML in recent years to solve problems in DB systems. The main thrust in this area of research has been in developing ML models and algorithms, often using deep learning techniques, that empirically outperform existing methods in real world DB systems. We complement this by providing a formal framework to establish the learnability of the selectivity estimation problem. As "ML for DB" advances further, we hope that the formal lens that we introduce in this paper can be adapted and generalized to a broader class of DB problems.

Our Contributions. First, we formalize the learnability of the selectivity-estimation problem. Recall that a database is a collection of tuples, and a selection query is a predicate that selects a subset of these tuples. The selectivity of a selection query is the probability that a randomly selected tuple satisfies the query. In order to learn the selectivity function, we employ the agnostic-learning framework [16], an extension of the classical PAC learning framework for real-valued functions, where we are given a set of sample queries and their respective selectivities from a fixed distribution (the *training set*), and our goal is to design an algorithm that can output the selectivity of a new query from the same distribution with high accuracy (see Figure 1 for an example).

Classical PAC learning theory asserts that a Boolean function is learnable if its VC-dimension is bounded. Generalizing this notion, it has been shown that a real-valued function is learnable using finitely many samples if its *fat shattering dimension* (defined in Section 2) is bounded [4, 6, 20]. This reduces the question of learnability of selectivity functions to bounding their respective fat shattering dimensions. We further note that selectivity functions correspond to selection queries on the underlying data. Each selection query, in turn, is a binary function on the data (i.e., which data items satisfy the query predicate), and the complexity of a class of binary functions is captured by its *VC-dimension* [45]. Our main result shows that if a class of selection queries has bounded

.

VC-dimension, then the fat shattering dimension of the corresponding selectivity function must also be bounded, and therefore, the selectivity function for such queries is learnable.

This result has several implications for important query classes:

- **Orthogonal Range Queries.** Such queries are specified as a conjunction of range conditions on individual attributes, e.g.:

  SELECT * FROM T WHERE $a_1 \leq A_1 \leq b_1$ AND $a_2 \leq A_2 \leq b_2$

  They are widely used as building blocks in more complex queries, and their selectivity-estimation (even for the simplest 1D range queries involving just a single attribute) has been the bread and butter of cost-based query optimizers, which uses selectivity estimates to gauge the intermediate result sizes and choose low-cost query execution plans. Taking a geometric view, we can represent each data tuple defined on d attributes as a point in $\mathbb{R}^d$ and each query as a *hyper-rectangle* in the same space. Known bounds on the VC-dimension of hyper-rectangles [21] then allow us to conclude that their selectivity is learnable.
- **Linear Inequality Queries.** Such queries allow multiple attributes to be brought together into one linear inequality, e.g.:

  SELECT * FROM T
  WHERE $\theta_0 + \theta_1 \times A_1 + \theta_2 \times A_2 + \cdots + \theta_d \times A_d \geq 0$

  Able to capture more complex conditions that can encode data correlations, these queries are popular in advanced analytical data processing systems. As earlier, we can represent each data tuple on d attributes as a point in $\mathbb{R}^d$. Then, each query is a *half-space* in $\mathbb{R}^d$. Again, using known bounds on the VC-dimension of halfspaces [21], we can conclude that the corresponding selectivity function is learnable.
- **Distance-based Queries.** These queries specify a "reference" object and find all objects that within some distance of it, e.g.:

  SELECT * FROM T
  WHERE $(A_1 - a_1)^2 + (A_2 - a_2)^2 + \cdots + (A_d - a_d)^2 \leq r^2$

  Here the reference object is $(a_1, \ldots, a_d)$ and the Euclidean (ℓ_2) distance threshold is r. Such queries have broad applications in text and image search, product recommendations, database optimization, network traffic, etc. Again, selectivity estimation enables cost-based optimization of queries involving such constructs. Moreover, the estimates may be of interest themselves; e.g., we might be interested in just counting how many other objects are in the vicinity of one object. As before, we use a geometric view where the data points are in $\mathbb{R}^d$ for d attributes, and the above query is a d-dimensional ℓ_2-ball. Invoking the standard bound [21] on the VC-dimension of ℓ_2-balls, we can conclude that the corresponding selectivity function is learnable.

While our framework establishes the learnability of the selectivity of above query types from a small set of training examples, it does not by itself prescribe any specific model or learning algorithm. As part of establishing the learnability of our selectivity query, we also need a procedure that, given a set of training samples and a family of data distributions (e.g. histograms, discrete distributions), constructs a data distribution from the given family that "best fits" the training samples. Our framework then guarantees that the learned data distribution estimates the selectivity of any query chosen from the same distribution as the training samples with high accuracy. For specific query types (e.g., orthogonal range queries), there already exists a large body of work on the selectivity-estimation problem, and our framework now gives them a solid foundation. To demonstrate the power of our framework beyond justifying existing methods, we further propose a simple, generic approach that embodies our theoretical results, and empirically validates its efficiency using extensive experiments. It is important to note that we are not designing this generic approach to "beat" existing methods with novel or sophisticated features; in fact, we intentionally avoid sophisticated features so that experimental comparison can focus on illustrating the power of our unifying framework instead of the artifacts of extra features. Despite the simplicity of our approach, our experimental results show that it performs comparably to the state-of-the-art methods for orthogonal range queries. Furthermore, for query classes that have seen less previous research, such as linear inequality and distance-based queries, our generic approach also work effectively, demonstrating the generality of the our theoretical framework.

Roadmap. This paper is organised as follows. In Section 1, we introduce the background knowledge of range query and selectivity estimation. In Section 2, we focus on the statistical learning question of determining the sample complexity of training for selectivity estimation problem under the agnostic-learning framework. In Section 3, we propose two simple generic algorithms for computing a data distribution that minimizes the expect loss function on a finite set of training queries. In Section 4, we implement these two algorithms to verify our theory, both of which are trained using a certain number of queries for obtaining small predication error on test queries, and compare them with state-of-art methods under the same framework.

2 LEARNABILITY OF QUERY SELECTIVITY

A *range space* Σ is a pair $(X, \mathcal{R})$, where X is a set of *objects* and $\mathcal{R}$ is a collection of subsets of X called *ranges*. For example, $X = \mathbb{R}^d$ and $\mathcal{R}$ can be the set of all d-dimensional rectangles, halfspaces, or balls. Let D be a probability distribution over X. For a given D, we define the *selectivity function* $s_D : \mathcal{R} \to [0, 1]$ as

$$s_D(R) = \Pr_{x \sim D}[x \in R].$$

Our goal is to *learn* the selectivities of the ranges in a range space Σ under an unknown data distribution from a finite sample of ranges and their respective selectivities. Formally, we define this learning task as follows.

2.1 The Learning Framework

Learnability. Following the agnostic learning model proposed by Haussler [16] (see also [4, 6]), which generalizes the PAC model, we define learnability in a more general setting. Let $\mathscr{H}$ be a family of functions from a domain Y to $[0, 1]$. Set $Z = Y \times [0, 1]$. For a function $H \in \mathscr{H}$, we define the *loss function* $\ell_H : Z \to [0, 1]$. For $z = (y, w) \in Z$,

$$\ell_H(z) = (H(y) - w)^2 .$$

For a probability distribution Q over Z and for a function $H \in \mathscr{H}$, we define

$$\mathrm{er}_Q(H) = \int_Z \ell_H(z) dQ(z) \tag{1}$$

to be the mean square loss of H with respect to distribution Q.

A *learning procedure* $\mathcal{A}$ is mapping from finite sequences in Z to $\mathscr{H}$. Given a *training sample* $\mathbf{z}^n = (z_1, z_2, \cdots, z_n) \in Z^n$, $\mathcal{A}$ returns a function $\mathcal{A}(\mathbf{z}^n)$. Given $\epsilon, \delta \in (0,1)$ and an integer $n > 0$, we say that $\mathcal{A}$ (ϵ, δ)-learns (agnostically) from n random training samples with respect to $\mathscr{H}$ if

$$\sup_Q \Pr[\mathrm{er}_Q(\mathcal{A}(\mathbf{z}^n)) \geq \inf_{H \in \mathscr{H}} \mathrm{er}_Q(H) + \epsilon] \leq \delta,$$

where Pr denotes the probability with respect to a random sample $\mathbf{z}^n \in Z^n$, each of $z_1, z_2, \cdots, z_n$ is drawn independently from Z at random according to Q, and supremum is taken over all distributions defined on Z. For $\epsilon > 0$, $\mathscr{H}$ is called *ϵ-learnable* if there exists a function $n_0 : [0,1]^2 \to \mathbb{N}$ and a learning procedure $\mathcal{A}$ such that for all $\delta > 0$ and for all $n \geq n_0(\epsilon, \delta)$, $\mathcal{A}$ (ϵ, δ)-learns from n examples with respect to $\mathscr{H}$; $n_0(\epsilon, \delta)$ is referred to as the minimum training set size for $\mathscr{H}$. Finally, $\mathscr{H}$ is *learnable* if it is ϵ-learnable for all $\epsilon > 0$.

VC dimension. Returning to the selectivity function of range space $\Sigma = (X, \mathcal{R})$, let $\mathscr{D}$ be a set of distributions defined on X. Set $\mathcal{S}_{\Sigma,\mathscr{D}} = \{s_D \mid D \in \mathscr{D}\}$, a family of functions from $\mathcal{R}$ to $[0,1]$. Set $Z = \mathcal{R} \times [0,1]$. Our main result is a characterization of learnability of $\mathcal{S}_{\Sigma,\mathscr{D}}$ in terms of the VC-dimension of Σ, defined below.

A subset $P \subseteq X$ is *shattered* by $\mathcal{R}$ if $\{P \cap R \mid R \in \mathcal{R}\} = 2^P$. The *VC-dimension* of $\mathcal{R}$, denoted by $\mathsf{VC\text{-}dim}(\Sigma)$, is the size of the largest subset of X that can be shattered by Σ. An example is given in Figure 2. If the VC-dimension of Σ is not bounded by a constant, then $\mathsf{VC\text{-}dim}(\Sigma) = \infty$. Our main result, stated in the theorem below, is that $\mathcal{S}_{\Sigma,\mathscr{D}}$ is learnable if and only if $\mathsf{VC\text{-}dim}(\Sigma)$ is finite.

Theorem 2.1. *Let $\Sigma = (X, \mathcal{R})$ be a range space, let $\mathscr{D}$ be a set of distributions defined on X, and let $\epsilon \in (0,1)$ be a parameter. If $\mathsf{VC\text{-}dim}(\Sigma) = \lambda$, for some constant $\lambda > 0$, then the family $\mathcal{S}_{\Sigma,\mathscr{D}}$ of selectivity functions is ϵ-learnable with a training set of size $\tilde{O}\left(\frac{1}{\epsilon^{\lambda+3}}\right)$.*[1] *Conversely, if $\mathsf{VC\text{-}dim}(\Sigma) = \infty$, $\mathcal{S}_{\Sigma,\mathscr{D}}$ is not (agnostically) learnable.*

Remark. Note that we do not assume training sample $z_i = (R_i, s_i) \in Z$ to be of the form $s_i = s_D(R_i)$ for some data distribution $D \in \mathscr{D}$. They are drawn from some distribution Q defined on $\mathcal{R} \times [0,1]$, and the goal is to learn the selectivity function in $\mathcal{S}_{\Sigma,\mathscr{D}}$ that minimizes the mean square loss. This is important, which allows us to decouple training samples from the family of functions, and the problem just becomes to find a function from the given family that minimizes the expected loss. This model is more general than the one assuming training sample in a form of $z_i = (R_i, s_D(R_i))$ for some data distribution $D \in \mathscr{D}$, for example, capturing the noisy input for learning the selectivity functions.

Instead of using the mean square error in (1), we can use other loss functions such as the L_1-norm or L_∞-norm of the error, i.e., $\int_{(y,w)} |H(y) - w| dQ(y,w)$ or $\sup_{(y,w)} |H(y) - w|$. Furthermore, the theorem holds for any $\mathscr{D}$, the family of data distributions and the bound on the training size is independent of $\mathscr{D}$. It might be possible to obtain an improved bound on the training size for certain family of data distributions. Finally, the theorem assumes the existence of a procedure that efficiently computes the function in $\mathcal{S}_{\Sigma,\mathscr{D}}$ that minimizes, or minimizes within additive error ϵ, the mean square loss over the finite sequence of training samples; see Section 3.

[1] $\tilde{O}(.)$ to hide lower order terms that are in polylog $\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ for constant λ.

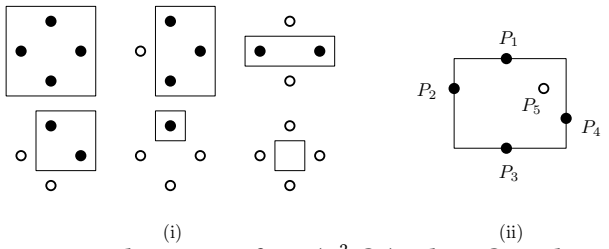

Figure 2: VC-dimension of $\Sigma = (\mathbb{R}^2, \mathcal{R}_\square)$, where $\mathcal{R}_\square$ is the set of all two-dimensional rectangles, is 4. (i) is an illustration of a set of 4 points shattered by $\mathcal{R}_\square$. On the other hand, no set $Y = \{p_1, p_2, p_3, p_4, p_5\}$ in $\mathbb{R}^2$ can be shattered by $\mathcal{R}_\square$ in (ii): let $\{P_1, P_2, P_3, P_4\} \subseteq Y$ be the subset of (at most 4) points of Y with extreme x- and y-coordinates. Then any rectangle containing P_1, P_2, P_3, P_4 also contains P_5.

2.2 Implications of Theorem 2.1

Before proving Theorem 2.1, we give some of its implications. We begin with the query classes mentioned in the introduction.

Orthogonal Range Queries: The range space $\Sigma_\square = (\mathbb{R}^d, \mathcal{R}_\square)$ for orthogonal range queries is defined as

$$\mathcal{R}_\square = \{\times_{i=1}^d [a_i, b_i] : a_i, b_i \in \mathbb{R}, a_i \leq b_i, \forall i \in [d]\}.$$

It is well known that $\mathsf{VC\text{-}dim}(\Sigma_\square) = 2d$ [21] (see Figure 2 for $d = 3$), therefore Theorem 2.1 implies that for any family $\mathscr{D}$ of distributions defined on $\mathbb{R}^d$ and for any $\epsilon > 0$, the selectivity functions are ϵ-learnable with training set of size $\tilde{O}\left(\frac{1}{\epsilon^{2d+3}}\right)$.

Linear Inequality Queries: The range space $\Sigma_\setminus = (\mathbb{R}^d, \mathcal{R}_\setminus)$ for linear inequality queries is defined as

$$\mathcal{R}_\setminus = \{R_{\setminus(a,b)} : a \in \mathbb{R}^d, b \in \mathbb{R}\},$$

where $R_{\setminus(a,b)} = \{x \in \mathbb{R}^d : a \cdot x \geq b\}$. It is known that $\mathsf{VC\text{-}dim}(\Sigma_\setminus) = d + 1$ [21], therefore Theorem 2.1 implies that for any family $\mathscr{D}$ of distributions defined on $\mathbb{R}^d$ and for any $\epsilon > 0$, the selectivity functions are ϵ-learnable with training set of size $\tilde{O}\left(\frac{1}{\epsilon^{d+4}}\right)$.

Distance-Based Queries: The range space $\Sigma_\circ = (\mathbb{R}^d, \mathcal{R}_\circ)$ for distance-based queries is defined as

$$\mathcal{R}_\circ = \{R_{\circ(a,b)} : a \in \mathbb{R}^d, b \in \mathbb{R}\},$$

where $R_{\circ(a,b)} = \{x \in \mathbb{R}^d : \|x - a\|_2 \leq b\}$ and $\|\cdot\|$ is the Euclidean norm. It is known that $\mathsf{VC\text{-}dim}(\Sigma_\circ) \leq d + 2$ [21], therefore Theorem 2.1 implies that for any family $\mathscr{D}$ of distributions defined on $\mathbb{R}^d$ and for any $\epsilon > 0$, the selectivity functions are ϵ-learnable with training set of size $\tilde{O}\left(\frac{1}{\epsilon^{d+5}}\right)$.

Semi-algebraic Range Queries. A very general class of range queries is the so-called *semi-algebraic range query*. A d-dimensional semi-algebraic set is subset of $\mathbb{R}^d$ defined by a Boolean formula over

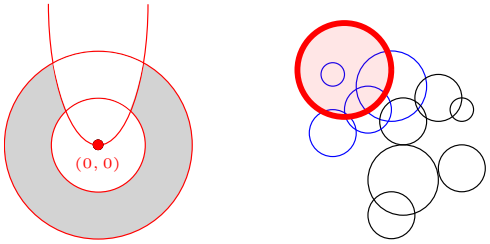

Figure 3: (Left) Semi-algebraic range of $R = \{(x, y) \in \mathbb{R}^2 \mid (x^2 + y^2 \leq 4) \wedge (x^2 + y^2 \geq 1) \wedge (y - 2x^2 \leq 0)\}$. (Right) A disc-intersection query, discs intersected by the query disc (red) are shown in blue.

polynomial inequality. For example, $R = \{(x, y) \in \mathbb{R}^2 \mid (x^2 + y^2 \leq 4) \wedge (x^2 + y^2 \geq 1) \wedge (y - 2x^2 \leq 0)\}$ is a semi-algebraic sets; see Figure 3. All the three above examples are special cases of semi-algebraic range queries. Let $\mathbb{T}_{d,b,\Delta}$ be the set of all semi-algebraic sets defined by at most b d-variate polynomial inequalities, each of degree at most Δ. It is known that the VC-dimension of range space $(\mathbb{R}^d, \mathbb{T}_{d,b,\Delta})$ is a constant $\lambda := \lambda(d, b, \Delta)$ [8]. Hence the selectivity functions on $(\mathbb{R}^d, \mathbb{T}_{d,b,\Delta})$ are also learnable for any constants d, b, Δ.

Semi-algebraic sets enable us to handle range spaces in which X is not a set of points in $\mathbb{R}^d$. For example, let $\mathbb{B}$ be the set of all discs in $\mathbb{R}^2$. For a query disc B, let $R_B \subseteq \mathbb{B}$ be the set of discs that intersect B; see Figure 3. Define $\mathcal{R}_\bullet = \{R_B \mid B \in \mathbb{B}\}$, and consider the range space $\Sigma_\bullet = (\mathbb{B}, \mathcal{R}_\bullet)$. We can map each disc in $\mathbb{B}$ to a point (x, y, z) in $\mathbb{R}^3$ where (x, y) is the center of the disc and z is its radius. Then for a query disc B centered at (c_x, c_y) and radius r, the range R_B maps to the set

$$\gamma_B = \{(x, y, z) \in \mathbb{R}^3 \mid (x - c_x)^2 + (y - c_y)^2 \leq (r + z)^2, z \geq 0\}.$$

Set $\mathbb{R}^3_{z\geq 0} = \mathbb{R}^2 \times \mathbb{R}_{z\geq 0}$ and $\hat{\mathcal{R}}_\bullet = \{\gamma_B \mid B \in \mathbb{B}\}$. Then $\Sigma_\bullet$ is mapped to $(\mathbb{R}^3_{z\geq 0}, \hat{\mathcal{R}}_\bullet)$. Since ranges in $\hat{\mathcal{R}}$ are semi-algebraic sets with $b = 1$ and $\Delta \leq 2$, VC-dim$(\mathbb{R}^3_{\geq 0}, \hat{\mathcal{R}}_\bullet)$ is finite and hence selectivity functions on $(\mathbb{B}, \mathcal{R})$ are learnable.

We conclude this discussion by giving an example of range space for which selectivity functions are not learnable.

Polygon range queries with arbitrary number of vertices. Let $\mathbb{C}$ be the set of all convex polygons in $\mathbb{R}^2$ with arbitrary number of vertices. Consider the range space $\Sigma = (\mathbb{R}^2, \mathbb{C})$. It is known that VC-dim$(\Sigma) = \infty$ [17], therefore Theorem 2.1 implies that selectivity functions on Σ are not learnable.

2.3 Proof of Theorem 2.1

We prove Theorem 2.1 using the notion of *fat-shattering dimension* introduced by Kearns and Schapire [20], which is a generalization of VC-dimension, and the results by Alon et al. [4] and Bartlett-Long [6] (see also [7]). As in Section 2.1, let $\mathscr{H}$ be a class of functions from a domain X into $[0, 1]$. Let $\gamma \in (0, 1/2)$ be a parameter. We say that $\mathscr{H}$ *γ-shatters* a subset $V \subseteq X$ if there is a witness function $\sigma : V \to [0, 1]$ such that for every subset $E \subseteq V$, there is a function $H_E \in \mathscr{H}$ with

$$\begin{aligned} H_E(x) &\geq \sigma(x) + \gamma, \quad \forall x \in E, \\ H_E(x) &\leq \sigma(x) - \gamma, \quad \forall x \in V \setminus E. \end{aligned} \tag{2}$$

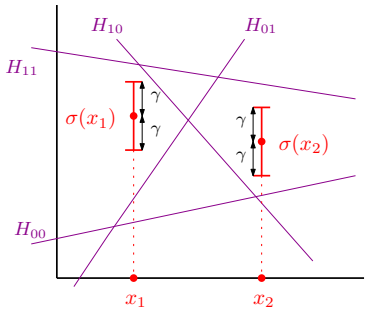

Figure 4: x_1, x_2 are γ-shattered by linear functions. we choose H to be the linear function whose bit sequence $b_2 b_1$ corresponds to E (i.e., $b_i = 1$ if $x_i \in E$).

An example is shown in Figure 4.

The *γ-fat shattering dimension* of $\mathscr{H}$, denoted by $\text{fat}_{\mathscr{H}}(\gamma)$, is the size of the largest subset of X that can be γ-shattered by $\mathscr{H}$. If subsets of unbounded finite size can be γ-shattered by $\mathscr{H}$, then we set $\text{fat}_{\mathscr{H}}(\gamma) = \infty$. Note that if $\mathscr{H}$ is a class of functions from X into $\{0, 1\}$, then γ-fat shattering dimension is the same as VC-dimension. An advantage of γ-fat shattering dimension is that it is sensitive to the scale at which difference in the function values are considered important. Alon et al. [4] proved that if $\text{fat}_{\mathscr{H}}(c\epsilon)$ is finite, where $c \in (0, 1)$ is a suitable constant, then $\mathscr{H}$ is ϵ-learnable. The bound on the size of the training set was improved by Bartlett and Long [6]. In particular, their result implies that $\mathscr{H}$ is ϵ-learnable with training-set size

$$n_0(\epsilon, \delta) = O\left(\frac{1}{\epsilon^2}\left(\text{fat}_{\mathscr{H}}\left(\frac{\epsilon}{9}\right)\log^2\frac{1}{\epsilon} + \log\frac{1}{\delta}\right)\right).$$

Returning to the selectivity functions, let $\Sigma = (X, \mathcal{R})$ be a range space, let $\mathscr{D}$ be a family of probability distributions on X and $\gamma \in (0, 1)$. Set $\mathcal{S} := \mathcal{S}_{\Sigma,\mathscr{D}}$ to be the selectivity functions defined by $\mathscr{D}$. Our main technical result is that if VC-dim$(\Sigma) = \lambda$, for some constant λ, then $\text{fat}_{\mathcal{S}}(\gamma) = \tilde{O}\left(\frac{1}{\gamma^{\lambda+1}}\right)$. By plugging this result into the results of [4, 6], we prove the first part of Theorem 2.1.

Let $\mathcal{T} \subseteq \mathcal{R}$ be a subset γ-shattered by $\mathcal{S}$. To bound $\text{fat}_{\mathcal{S}}(\gamma)$, it suffices to prove that $|\mathcal{T}| = \tilde{O}\left(\frac{1}{\gamma^{\lambda+1}}\right)$. First, we partition the ranges in $\mathcal{T}$ based on the values of their respective witnesses $\sigma(R)$[2]:

$$\mathcal{T}_j = \{R \in \mathcal{T} : \sigma(R) \in [(j-1)\cdot\gamma, j\cdot\gamma], \text{for} j \in [1/\gamma]\}.$$

Lemma 2.2. *Suppose Equation (2) is realized for some subset $E \in \mathcal{T}_j$ by s_D for some distribution $D \in \mathscr{D}$. Then, for any pair $R \in E, R' \in \mathcal{T}_j \setminus E$, we have*

$$s_D(R) - s_D(R') > \gamma. \tag{3}$$

Proof. By Equation (2), we have

$$s_D(R) \geq \sigma(R) + \gamma \quad \text{and} \quad -s_D(R') \geq -\sigma(R') + \gamma$$

Adding these, we get

$$(s_D(R) - s_D(R')) + (\sigma(R') - \sigma(R)) \geq 2\gamma. \tag{4}$$

[2]Note that although $\sigma(R) = 1$ is excluded by this definition if $1/\gamma$ is an integer, it is a well-defined partition since $\sigma(R)$ cannot be equal to 1 for any range $R \in \mathcal{T}$. This follows from the observation that if $\sigma(R) = 1$, then Equation (2) cannot be satisfied for $R \in E$ since $H_E(R) \leq 1$ and $\gamma > 0$.

Since $R, R' \in \mathcal{T}_j$ for some $j \in \lceil 1/\gamma \rceil$, we have $\sigma(R') - \sigma(R) < \gamma$. The lemma follows by using this inequality in Equation (4). □

Now, consider any fixed ordering $\pi = \langle R_1, R_2, \cdots, R_k \rangle$ of the ranges in $\mathcal{T}_j$, where $k = |\mathcal{T}_j|$. Let us also fix the subset:

$$E = \{R_{2i} \mid 1 \leq i \leq \lfloor k/2 \rfloor\} \tag{5}$$

to be the set of ranges with even index in π. We say that an object $x \in X$ crosses a pair of ranges R, R' if $x \in R \oplus R'$, where $\oplus$ is the symmetric difference (see Figure 1 for $R_1 \oplus R_3$). For $1 \leq i < k$ and for every $x \in X$, we define an indicator random variable as follows:

$$I_{i,x} = \begin{cases} 1 & \text{if } x \in R_i \oplus R_{i+1}, \\ 0 & \text{otherwise,} \end{cases}$$

and let $I_x = \sum_{i=1}^{k-1} I_{i,x}$.

Since $\mathcal{T}$ is γ-shattered by $\mathcal{S}$, there is a distribution $D_\pi \in \mathcal{D}$ that satisfies (2) for E. The next lemma is a direct consequence of Lemma 2.2, by summing up over the pairs of ranges R_i, R_{i-1} for even i in $\mathcal{T}_j$:

Lemma 2.3. $\mathbb{E}_{x \sim D_\pi}[I_x] > \gamma(k-1)$.

Proof. By Lemma 2.2, $\mathbb{E}_{x \in D_\pi}[I_{i,x}] \geq \gamma$ for any index i since exactly one of R_i, R_{i+1} belongs to E. The lemma now follows by using linearity of expectation. □

The lower bound on $\mathbb{E}_{x \sim D_E}[I_x]$ in Lemma 2.3 holds for any ordering π of the ranges in $\mathcal{T}_j$; the distribution D_π obviously depends on π. We now complement this lower bound with an upper bound on $\mathbb{E}_{x \sim D_\pi}[I_x]$ for a *specific* ordering π of $\mathcal{T}_j$.

Lemma 2.4. *There is an ordering $R_1, R_2, \cdots, R_k$ of the ranges in $\mathcal{T}_j$ such that for any distribution D defined on X, we have:*

$$\mathbb{E}_{x \in D}[I_x] = O(k^{1-1/\lambda} \log k),$$

where $\lambda = \mathsf{VC\text{-}dim}(X, \mathcal{R})$.

Proof. Let $\widetilde{\Sigma} = (X, \mathcal{T}_j)$ be the range space defined by the ranges in $\mathcal{T}_j$. Note that $\mathsf{VC\text{-}dim}(\widetilde{\Sigma}) \leq \mathsf{VC\text{-}dim}(\Sigma) = \lambda$. Consider the dual range space $\widetilde{\Sigma}^*$ of $\widetilde{\Sigma}$, where $\widetilde{\Sigma}^* = (\mathcal{T}_j, \{\mathcal{R}_x = \{R \in \mathcal{T}_j : x \in R\} \mid x \in X\})$, i.e., the objects of $\widetilde{\Sigma}^*$ are the ranges of $\mathcal{T}_j$ and for each object $x \in X$, we have a dual range in $\widetilde{\Sigma}^*$ consisting of ranges of $\widetilde{\Sigma}$ that contain x. Note that $\widetilde{\Sigma}^{**} = \widetilde{\Sigma}$.

We compute the desired ordering of $\mathcal{T}_j$, using the following results by Chazelle and Welzl [11]: Let $\Xi = (V, \Gamma)$ be a finite range space with $|V| = m$. We say that a range $\gamma \in \Gamma$ crosses a pair $v_i, v_j \in V$ if $|\gamma \cap \{v_i, v_j\}| = 1$. The result in [11] (Theorem 4.3) proves that there is an ordering $v_1, v_2, \cdots, v_m$ of objects in V such that any range in $\mathcal{T}$ crosses $O\left(m^{1-1/\lambda^*} \log m\right)$ pairs (v_i, v_{i+1}) for $1 \leq i < m$, where λ^* is the VC-dimension of the dual range space of Ξ.[3] Applying this result to $\widetilde{\Sigma}^*$ and using the fact that $\widetilde{\Sigma}^{**} = \widetilde{\Sigma}$, we obtain an ordering $R_1, R_2, \cdots, R_k$ of $\mathcal{T}_j$ such that any range of $\widetilde{\Sigma}^*$ crosses $O\left(k^{1-1/\lambda} \log k\right)$ pairs (R_i, R_{i+1}). By the definition, a range $\mathcal{R}_x$ crosses R_i, R_{i+1} if $|\mathcal{R}_x \cap \{R_i, R_{i+1}\}| = 1$, which is equivalent to saying that $x \in R_i \oplus R_{i+1}$. Hence, for any $x \in X$, there are $O\left(k^{1-1/\lambda} \log k\right)$ pairs (R_i, R_{i+1}) crossed by x. Since this bound holds for every $x \in X$, we conclude that

$$\mathbb{E}_{x \sim D}[I_x] = O\left(k^{1-1/\lambda} \log k\right). \qquad \square$$

We are now ready to bound the size of $\mathcal{T}_j$.

Lemma 2.5. *For any $j \in \lceil 1/\gamma \rceil$, $|\mathcal{T}_j| = O\left((\frac{1}{\gamma} \log \frac{1}{\gamma})^\lambda\right)$.*

Proof. Plugging Lemmas 2.4 and 2.3 together, we conclude there exists a constant c such that

$$\gamma \cdot (k-1) \leq c \cdot k^{1-1/\lambda} \log k,$$

which implies that $\frac{k^{1/\lambda}}{\log k} \leq 2c/\gamma$, or $k = O\left((\frac{1}{\gamma} \log \frac{1}{\gamma})^\lambda\right)$. □

Summing this bound over all $j \in \lceil 1/\gamma \rceil$, we conclude that $|\mathcal{T}| = \tilde{O}\left(\frac{1}{\gamma^{\lambda+1}}\right)$. Hence, the size of any set of query ranges in $\mathcal{R}$ that can be γ-shattered by $\mathcal{S}$ is $\tilde{O}\left(\frac{1}{\gamma^{\lambda+1}}\right)$, which implies the main technical result of this section.

Lemma 2.6. *Let $\Sigma = (X, \mathcal{R})$ be a range space with $\mathsf{VC\text{-}dim}(\Sigma) = \lambda$, let $\mathcal{D}$ be a family of probability distribution over X, and let $\mathcal{S} := \mathcal{S}_{\Sigma,\mathcal{D}}$ be the family of selectivity functions on Σ by $\mathcal{D}$. For any $\gamma \in (0,1)$, the γ-fat shattering dimension of $\mathcal{S}$ is $\tilde{O}\left(\frac{1}{\gamma^{\lambda+1}}\right)$.*

Finally, plugging Lemma 2.6 into the results of Alon et al. [4] and Bartlett-Long [6], we obtain the first part of Theorem 2.1.

We next turn to the second part of Theorem 2.1. As in Section 2.1, let $\mathscr{H}$ be a class of functions from a domain X into $[0,1]$. Let $\gamma \in [0,1]$ be a parameter. Alon et al. [4] proved that if $\mathrm{fat}_{\mathscr{H}}(\epsilon) = \infty$, then $\mathscr{H}$ is not $(\epsilon^2/8 - \tau)$-learnable for any $\tau > 0$. Returning to the selectivity functions $\mathcal{S} := \mathcal{S}_{\Sigma,\mathcal{D}}$ defined on the range space $\Sigma = (X, \mathcal{R})$ and a family of probability distribution on X as $\mathcal{D}$. Our second technical result is that if $\mathsf{VC\text{-}dim}(\Sigma) = \infty$, then $\mathrm{fat}_{\mathcal{S}}(\gamma) = \infty$ for any $\gamma \in (0, 1/2)$.

Lemma 2.7. *Let $\Sigma = (X, \mathcal{R})$ be a range space, let $\mathcal{D}$ be a family of probability distribution over X, and let $\mathcal{S} := \mathcal{S}_{\Sigma,\mathcal{D}}$ be the family of selectivity functions on Σ by $\mathcal{D}$. If $\mathsf{VC\text{-}dim}(\Sigma) = \infty$, the γ-fat shattering dimension of $\mathcal{S}$ is also ∞, for any $\gamma \in (0, 1/2)$.*

Proof. Consider the dual range space Σ^* of Σ, where $\Sigma^* = (\mathcal{R}, \Gamma)$ where $\Gamma = \{\mathcal{R}_x = \{R \in \mathcal{R} : x \in R\} \mid x \in X\}$ as defined in the proof of Lemma 2.4. As shown in [11], since $\mathsf{VC\text{-}dim}(\Sigma) = \infty$, we have $\mathsf{VC\text{-}dim}(\Sigma^*) = \infty$. In other words, for any integer $k > 0$, there exists a subset $\mathcal{T}_k \subseteq \mathcal{R}$ of k ranges shattered by Γ, i.e., for every subset $E \subseteq \mathcal{T}_k$, there is a point $x_E \in X$ such that $x_E \in R_i$ if all $R_i \in E$ and $x_E \notin R_i$ for all $R_i \in \mathcal{T}_k \setminus E$.

Next, we show that $\mathcal{T}_k$ is γ-shattered by $\mathcal{S}$. Set $\sigma(R_i) = 1/2$ for all $R_i \in \mathcal{T}_k$. Consider an arbitrary subset $E \subseteq \mathcal{T}_k$. We choose $D \in \mathcal{D}$ as a delta function, which is 1 at x_E and 0 everywhere else. The corresponding selectivity function $s_D \in \mathcal{S}$ has $s_D(R_i) = 1$ if $x_E \in R_i$ and 0 otherwise, which realizes Equation (2) for any $\gamma \leq 1/2$. Hence for any $k > 0$, there always exists a subset $\mathcal{T}_k \subseteq \mathcal{R}$ of size k that can be γ-shattered by $\mathcal{S}$ for any $\gamma \in (0, 1/2)$, i.e., the γ-fat shattering dimension of $\mathcal{S}$ is ∞. An example is illustrated in Figure 5. □

The above lemma proves second part of Theorem 2.1, thereby completing the proof of Theorem 2.1.

[3] For $\lambda^* = 1$, the original paper [11] proves a slightly weaker bound of $O(\log^2 m)$ on the number of pairs crossed by a range. Using an improved bound on ϵ-nets for range spaces of VC-dimension 1 (see e.g. [35], Chapter 15), the bound can be improved to $O(\log m)$.

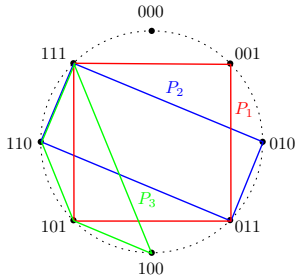

Figure 5: An example of 3 convex polygons P_1, P_2, P_3 that are γ-shattered for any $\gamma \in (0, 1/2)$. To satisfy Equation (2) for a subset $E \subseteq \{P_1, P_2, P_3\}$, we choose D_E to be the unit mass at the point whose bit sequence $b_3b_2b_1$ corresponds to E (i.e., $b_i = 1$ if $P_i \in E$).

3 LEARNING ALGORITHM

Recall that Theorem 2.1 gives an upper bound on the size of training samples, but the definition of ϵ-learnability assumes the existence of a learning procedure that for a given a finite training sample $\mathbf{z}^n = \{z_1, z_2, \cdots, z_n\}$ where $z_i = (R_i, s_i) \in \mathcal{R} \times [0, 1]$, and a family $\mathcal{D}$ of data distributions, computes a distribution $D \in \mathcal{D}$ such that s_D minimizes the expected loss function, i.e., it returns $\arg\min_{D \in \mathcal{D}} \frac{1}{n} \sum_{i=1}^{n} (s_D(R_i) - s_i)^2$. In this section, we describe algorithms for computing such a distribution. For simplicity, we focus on selectivity queries discussed in the introduction, namely orthogonal range, linear inequality, and distance-based queries, though our algorithm works for a much larger class of queries such as semi-algebraic range queries. The aim of this section is to describe simple, generic approaches, and we do not attempt to optimize the learning procedure for specific selectivity queries.

3.1 A Generic Procedure

We focus on two families of distributions, histograms and discrete distributions. In the former, a distribution is a piecewise-constant function, i.e., $D = \{(B_1, w_1), \cdots, (B_m, w_m)\}$, where $\sum_{i=1}^{m} w_i = 1$. D has uniform density $\frac{w_i}{\mathrm{Vol}(B_i)}$ over each *bucket* B_i, where $\mathrm{Vol}(B_i)$ is the volume of B_i, and each $B_i \subseteq \mathbb{R}^d$ is a simple region of constant complexity homomorphic to a ball (e.g., boxes, simplices, etc), also called Tarski cells [48]. B_i's are pairwise disjoint and partition $\mathbb{R}^d$. For a query range R, $s_D(R)$ is defined as

$$s_D(R) = \sum_{j=1}^{m} \frac{\mathrm{Vol}(B_i \cap R)}{\mathrm{Vol}(B_i)} \cdot w_i \tag{6}$$

Intuitively, $\frac{\mathrm{Vol}(B_i \cap R)}{\mathrm{Vol}(B_i)}$ computes the fraction of the bucket B_i that intersects with the query region R. Note that we do not make any assumption on the ranges, which can be bounded or unbounded. Multiplying this fraction by w_i, $s_D(R)$ in essence makes the simple assumption that the data points within each cell are distributed uniformly. We note that when the range R can be represented with a simple function, such as an orthogonal range, a halfspace or a ball, the volume of R and its intersection with a bucket (as hyperrectangle) can be easily computed exactly. In general, the volume of a complex range can be estimated via MCMC sampling [14].

A discrete distribution also has a similar form $D = \{(B_1, w_1), \cdots, (B_m, w_m)\}$, but B_i's are a set of m points, which we also call *buckets*, in $\mathbb{R}^d$. As before $\sum_{i=1}^{m} w_i = 1$. For a query range R, $s_D(R)$ is now defined as

$$s_D(R) = \sum_{i=1}^{k} \mathbf{1}(B_i \in R) \cdot w_i \tag{7}$$

In both cases, the algorithm computes D in two phases. The first phase, called *bucket-selection*, constructs the set $\mathcal{B} = \{B_1, B_2, \cdots, B_m\}$ of buckets. The second phase, called *weight-estimation*, computes the weight w_i for each bucket B_i.

Bucket design. Let $\{R_1, R_2, \cdots, R_n\}$ be the set of ranges in the training set $\mathbf{z}^n$; here we treat each range as a geometric region defined by the query predicate (e.g. rectangles for orthogonal range queries, halfspaces for linear-inequality queries, balls for distance-based queries) rather than a subset of input objects. The *arrangement* of $\{R_1, R_2, \cdots, R_n\}$ is the partition of $\mathbb{R}^d$ into maximal connected regions so that each region lies in the same subset of ranges in $\{R_1, R_2, \cdots, R_n\}$. We further refine each region into small regions, called *cells*, so that each cell has constant complexity (i.e., constant number of vertices, edges, and faces that only depends on d) and its boundary is connected. It is known that such a decomposition of size $O(n^d)$ can be computed in $O(n^d \log n)$ time [3]. We choose $\mathcal{B}$, the set of buckets, to be the resulting set of cells. If we wish to construct a discrete distribution, we simply choose a random point in each cell, and these points form the bucket set $\mathcal{B}$.

Weight estimation Let $\mathcal{B}$ be the set of buckets constructed in the previous phase. To estimate the weights $w_1, w_2, \cdots, w_m$, we set them as variables and solve the following convex quadratic programming:

$$\begin{aligned} \textbf{minimize} \quad & \textstyle\sum_{i=1}^{n} (s_D(R_i) - s_i)^2 \\ \textbf{subject to} \quad & \textstyle\sum_{j=1}^{m} w_j = 1, \\ & 0 \le w_j \le 1, \quad j \in \{1, 2 \ldots, k\}. \end{aligned} \tag{8}$$

where $s_D(R_i)$ is the function specified in Equation (6) and (7) for histograms and discrete distributions respectively. We solve this problem using open-sourced *non-negative least squares* solver [1].

Lemma 3.1. *The above algorithm constructs a histogram (resp. discrete distribution) that minimizes the loss function over all histograms (resp. discrete distributions).*

Proof. We show that the discrete distribution D constructed by the above algorithm is optimal with respect the loss function. Suppose not, assume there exists a discrete distribution D^* that achieves smaller loss than D. Let $\mathcal{B}^*$ be the set of buckets in D^*. We construct another discrete distribution D' based on D^* as follows. For each cell c in the arrangement of $\{R_1, R_2, \cdots, R_n\}$, let $\mathcal{B}^*_c = \{B \in \mathcal{B}^* : B \in c\}$ be the set of buckets in $\mathcal{B}^*$ that falling inside c. We choose an arbitrary point inside c as B_c, and set its weight $w_c = \sum_{B \in \mathcal{B}^*} w(B)$, i.e., the sum of weights of buckets in $\mathcal{B}^*_c$ if $\mathcal{B}^*_c \neq \emptyset$, and $w_c = 0$ otherwise. It can be easily checked that D' achieves the same loss as D^*. Meanwhile, we observe that D' and D have the same set of buckets. This way, D achieves smaller at least not larger loss than D', implied by the optimality of (8). Together,

D achieves smaller smaller at least not larger loss than D^*, coming to a contradiction.

We next show that the histogram D constructed by the above algorithm is optimal with respect to the loss function, using a similar argument. By contradiction, assume there is a histogram D^* that achieves smaller loss than D. Let $\mathcal{B}^*$ be the set of buckets in D^*. We construct another histogram D' based on D^* as follows. For each cell c in the arrangement of $\{R_1, R_2, \cdots, R_n\}$, let $\mathcal{B}^*_c = \{B \in \mathcal{B}^* : B \cap c \neq \emptyset\}$ be the set of buckets in $\mathcal{B}^*$ that have non-empty intersection with c. We choose c as a bucket B_c, and set its weight $w_c = \sum_{B \in \mathcal{B}^*} \frac{\text{Vol}(B \cap c)}{\text{Vol}(c)} \cdot w(B)$, and $w_c = 0$ otherwise. It can be easily checked that D' achieves the same loss as D^*. Meanwhile, we observe that D' and D have the same set of buckets. This way, D achieves smaller at least not larger loss than D', implied by the optimality of (8). Together, D achieves smaller smaller at least not larger loss than D^*, coming to a contradiction. □

The main shortcoming of the above approach is that the complexity of the distribution depends on the training set and increases exponentially with dimension, in the worst case. Therefore, it is desirable to consider distributions with bounded complexity. For example, let $\mathscr{D}_k$ be the family of all histograms with at most k buckets where each bucket is a rectangle in $\mathbb{R}^d$, or the family of discrete distributions with support size at most k.

Given a training set $\mathbf{z}^n$, we are unaware of any polynomial-time algorithm for computing an optimal distribution of complexity k. The intractability of a number of related problems [32] suggests that the problem at hand is also NP-Hard, and we leave it as an interesting direction of future research. In the next two subsections, we describe simple, efficient algorithms for constructing a histogram and a discrete distribution. The weight-estimation phase remains the same, so we focus on the bucket-design phase.

3.2 Histogram

We construct a histogram QuadHist, intended for low-dimensional data and queries. For simplicity, we assume finite lower and upper bounds on the range of values for each dimension. Regardless of the query class—orthogonal range, linear inequality, or distance-based queries—QuadHist's buckets are a disjoint set of orthogonal ranges coming from the partitioning of D by a quadtree. The construction of the quadtree is guided by both the geometry of training queries and their selectivities, such that the resulting partitioning of the data space is finer in parts where queries and data are denser.

Let $\mathbf{z}^n = (z_1, z_2, \cdots, z_n)$ be the training set with $z_i = (R_i, s_i)$. We construct a quadtree on the ranges $R_1, R_2, \cdots, R_n$ in the training set $\mathbf{z}^n$ as follows. We start with a single-node quadtree corresponding to a single bucket spanning the whole data space. We process each $z_i = (R_i, s_i)$ to refine (if needed) the buckets as follows. For each leaf node B of the quadtree (interpreting B as a range), we compute $\frac{\text{Vol}(B \cap R_i)}{\text{Vol}(R_i)} \cdot s_i$. In the same spirit as $s_D(R)$, this quantity estimates the fraction (out of all data points) of the data points in R that are also in B. We compare this estimate with a predetermined threshold $\tau \in (0, 1)$. If the estimate is higher than τ (informally, B carries "too much" density), we split the quadtree leaf B into 2^d children and recursively apply the procedure on them. See Figure 6 for an illustration. After going through all training queries, we take all leaves of the final quadtree to be our QuadHist buckets (and proceed to the weight-assignment phase). We can control the model size k by varying the parameter τ or adding a hard termination condition on the number of leaves in the splitting procedure.

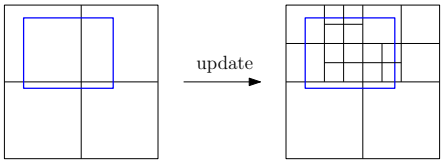

Figure 6: Bucket refinement for QuadHist. The blue rectangle is a training range R with selectivity 0.2 and the underlying grid is the quadtree leaves. Splitting is recursively applied to each node until the density of its intersection with R is estimated no greater than $\tau = 0.026$. The left is the quadtree before processing R, and the right is the quadtree after.

Remarks. Several points are worth noting here (details are in the Appendix A.1). Considering the selectivities in bucket design protects us from devoting more buckets than necessary to regions where data is sparse (although the weight estimation step utilizes selectivities, the buckets would have been chosen already).

Second, the simplicity of quadtree-guided bucket design procedure gives rise to an interesting and desirable property of *stability*: given a training workload, the resulting collection of buckets is always the same regardless of the ordering in which we process the workload. This property is unfortunately missing for many complex selectivity estimation schemes with more bells and whistles. Combining the stability of bucket design with the determinism of weight estimation, we know that QuadHist trained on the same query workload would always behave consistently.

Third, the quadtree doubles up as a convenient data structure for speeding up the bucket design step of the training process. For example, the τ-based splitting procedure can piggyback on the efficient and generic quadtree procedure for answering R as range query, regardless of R's shape.

3.3 Discrete Distribution

We present a discrete distribution PtsHist as an alternative instantiation of our generic approach for high dimensions. In high dimensions, QuadHist is not expected to perform well because of the well known challenges: 1) rectangles are poor representations of high-dimensional data distributions, and 2) computing volumes of intersections between orthogonal ranges and other types of query ranges (e.g., balls) in high dimensions is difficult. Hence, PtsHist turns to using a collection of *points* in the data space (as opposed to ranges) as buckets.

Given a target model size k, we take the following two steps to generate the points representing buckets. 1) We draw $0.9k$ points from the *interior* of all training query ranges. More specifically, for each $z_i = (R_i, s_i) \in \mathbf{z}^n$, we draw $\frac{s_i}{\sum_{j=1}^n s_j} \cdot (0.9k)$ points uniformly at random from the range defined by R. In other words, each R receives a "share" of points proportional to its selectivity. 2) We then draw the remaining $0.1k$ uniformly at random from the whole space.

This step essentially makes it possible to allocate some density to regions not covered by the training queries.

Although sampling from the interior of geometric objects in high dimensions has its own challenges, it is a well-studied problem for specific shapes such as hyperrectangles, halfspaces, and balls. Our sampling implementation in Section 4 in fact uses straightforward *rejection sampling* from the smallest bounding box [42] of R (see Appendix A.2 for details), and we have found the generic approach to offer adequate performance in practice. Figure 7 illustrates the real data distribution, the histogram by QuadHist, and the discrete distribution by PtsHist over the Power dataset (see Section 4).

Remarks. The sampling procedure used by PtsHist does not guarantee an unbiased sample from any data distribution D —but that is not our goal of this procedure in the first place. Instead, we only aim to generate a number of points whose positions serve as buckets; the subsequent (generic) weight estimation step ensures the consistency between the PtsHist model and the training workload.

4 EXPERIMENTS

In this section, we implement QuadHist and PtsHist in Section 3 as instantiations of our theoretical results in Section 2. We empirically evaluate their performance on real-world datasets and, when applicable, compare them against state-of-the-art solutions (for orthogonal range queries). As mentioned in Section 3, they are not intended to "beat" state-of-the-art solutions; rather, they are simple, generic implementations so that our experiments can focus on illustrating the power of our theoretical results instead of the artifacts of additional features. We implemented all our algorithms in Python and ran all our experiments on a server with 8 Intel Core i7-9700 CPUs (3.00GHz). All codes are public at [2].

Datasets [12]. We use real-world datasets adopted by a recent benchmark paper [46] for evaluation:

- **Power** contains electric power measurement gathered from a house over 47 months, with 2.1M tuples over 7 attributes.
- **Forest** contains forest cover type data, with 581k tuples over 10 numerical attributes. It is named as CoverType in [12].
- **Census** contains the basic population characteristics, with 49K tuples over 13 attributes (8 categorical and 5 numerical).
- **DMV** contains the vehicle registration records of NYC, with 11M tuples over 11 attributes (10 categorical and 1 numerical).

As datasets have multiple attributes, for a given experiment, we will choose a subset of attributes randomly and project the tuples on the chosen attributes. For simplicity, we normalize the domain of each attribute into $[0, 1]$.

Workloads. We consider orthogonal range queries, halfspace queries, and ball queries. For orthogonal range queries, we generate three different synthetic workloads. Each orthogonal range query R can be represented by a center point and d side lengths (one per dimension). After fixing the center point, we sample each side length independently and uniformly from $[0, 1]$. Depending on the distribution of centers points, we distinguish the following three workloads (illustrated in Figure 8):

- **Data-driven**: we generate centers by uniformly sampling from the underlying dataset.
- **Random**: we generate centers by uniformly sampling from the d-dimensional unit cube.
- **Gaussian**: we generate centers by uniformly sampling from a d-dimensional Gaussian distribution. We set the mean and variance for each dimension of the Gaussian distribution as 0.5 and 0.167.

The Data-driven workload is arguably more realistic as queries typically "follow" the underlying data distribution, but we also want to evaluate on Random and Gaussian, which are independent from the underlying data. We will only generate equality predicates for categorical attributes; hence the width is zero in this case.

Workloads for other query types are generated in an analogous fashion. For ball queries, once we pick the center point, we then sample the ball radius uniformly from $[0, 1]$. For halfspace queries, once we pick the center point (lying on the boundary plane of the halfspace), we then randomly pick a d-dimensional unit vector (normal to that plane) that defines the orientation of the halfspace.

Unless explicitly noted otherwise, the set of training and test queries for each experiment are sampled uniformly and independently from the same query workload. Note that there could be very few queries in the overlapping, as well as the subset of predicates.

Methods Compared. For fair comparison, we restrict ourselves to methods that only have access to query workload, but not the underlying data. Since this paper is concerned with learned selectivity estimation models that can provide provable guarantees, we also do not include methods based on deep learning that may return models that do not correspond to any valid hypothesis, and consequently, have been observed to produce selectivity estimates that are not monotone or consistent [46].

For orthogonal range queries, based on the recent empirical study on cardinality estimation in [46] (see Section 5 for more details), *Isomer* [39] produces the best accuracy and *QuickSel* [36] achieves the best tradeoff between accuracy and efficiency, so we include both in our comparison with QuadHist and PtsHist. For halfspace and ball queries, there are no obvious candidates for comparison with our methods, as traditional histogram-based methods have not focused on these queries.

Error Measures. We adopt two common error measures for evaluating selectivity estimators. For a test query R, let $\hat{s}(R)$ and $s(R)$ be the estimated and true selectivities of R, respectively, and let n be the number of test queries.

- **Root Mean Square (RMS) Error** $= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{s}(R_i) - s(R_i))^2}$.
- **Q-Error(p) [31]** = p-th quantile of $\left\{\frac{\max\{\hat{s}(R_i), s(R_i)\}}{\min\{\hat{s}(R_i), s(R_i)\}} : i \in [n]\right\}$.
- L_∞ **Error** $= \max_{i=1}^{n} |\hat{s}(R_i) - s(R_i)|$.

Q-error is a good complement of RMS error because Q-error is better at capturing errors that are small in absolute terms but large in relative terms, which occur frequently since many database queries tend to be selective. L_∞ error is used for investigating different objective functions in model training.

Outline. In the remaining of this section, we will investigate the following questions on the learned model for selectivity functions to verify our learning theory developed in Section 2:

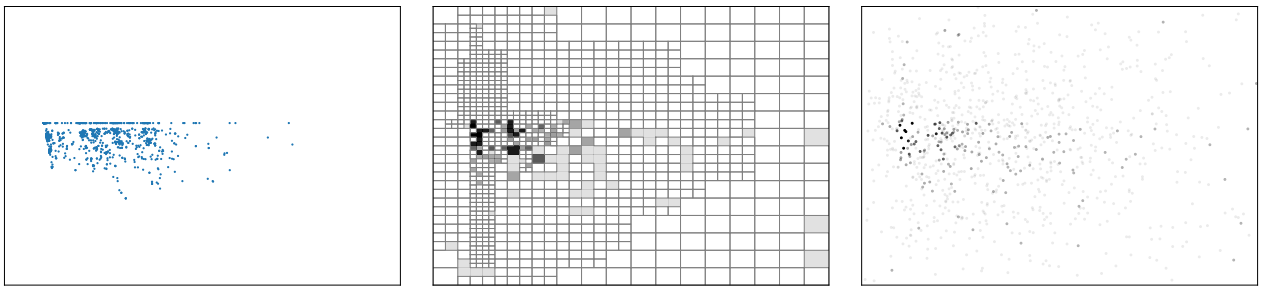

Figure 7: Illustration of underlying data distribution and our learned model. The left is the distribution of 1000 data points randomly drawn from Power dataset. The middle is the set of buckets of QuadHist constructed under threshold $\tau = 0.01$. The right is the set of buckets of PtsHist. Both QuadHist and PtsHist of size 1000 are built on 1000 training queries from the random workload of Power dataset. We darken the buckets with their associate weights.

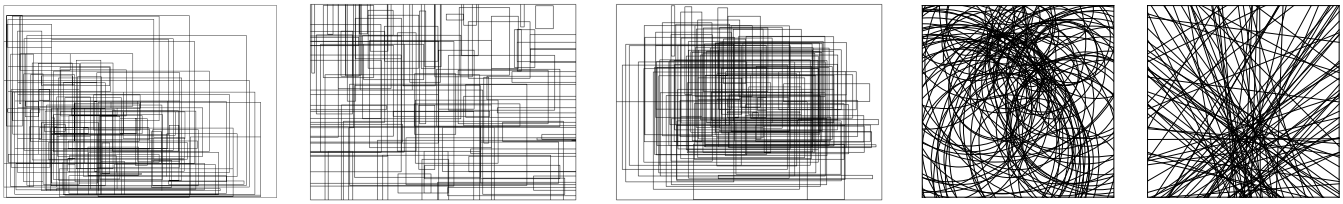

Figure 8: Illustration of workloads generated for Power dataset: (i) Data-driven workload for orthogonal range queries, (ii) Random workload for orthogonal range queries, (iii) Gaussian workload for orthogonal range queries, (iv) Data-driven workload for ball queries, (v) Data-driven workload for halfspace queries.

- Section 4.1: How does the training size affect the learned model?
- Section 4.2: What is the performance of learned model if the query workload is not correlated with the underlying data distribution?
- Section 4.3: What is the performance of learned model if training and testing query distributions do not match?
- Section 4.4: How does the dimensionality of data space affect the learned model?
- Section 4.5: How does the query type affect the learned model?
- Section 4.6: How does the objective function affect the learned model?

4.1 Learnability with Enough Training Samples

We start with selectivity estimation for orthogonal range queries, which has been studied extensively, and validate the learnability of selectivity functions. Recall that Theorem 2.1 shows a clear dependency between error ϵ and training size m: we now explore this dependency empirically. We first take a closer look at the 2D case, where QuadHist is our generic implementation. Figure 9 shows the QuadHist results for Data-driven query workload over Power data. We vary the number of training queries from 50 to 2000; each line plot corresponds to a specifically sized training set. Since the accuracy of QuadHist also depends on its model complexity (number of buckets), given a specific training set, we further vary τ in the bucket design step (Section 3.2) to adjust the model complexity. As we can see from Figure 9, error generally decreases with model complexity, although it eventually flattens out, and in one case even trends up because of overfitting (when using 10000 buckets for merely 50 training queries). As we increase the number of training queries, error decreases (the series of line plots push toward the origin), although the rate of decrease also diminishes, consistent with Theorem 2.1's prediction. The good news is that with 200 training queries and 500 buckets, QuadHist already offers practically acceptable accuracy with RMS error smaller than 0.02.

To help put our results in a boarder context, we next bring other state-of-the-art competitors, Isomer and QuickSel, into comparison. For completeness we also compare with our PtsHist (although it is intended for higher dimensions). The results are shown in Figures 10, 11, and 12. Again, we consider 2D orthogonal range queries for Data-driven query workload over Power data. As we have seen, model complexity can affect prediction accuracy, so for comparison, we adopt QuickSel's convention of using number of buckets 4× the number of training queries, for both QuadHist and PtsHist.[4] For Isomer, it is difficult to enforce this convention, so we let it choose the number of buckets by itself; in our results, Isomer ends up using number of buckets 48-160× the number of training queries.

As we vary the number of the training queries, Figure 10 shows the actual number of buckets used by different models; Figure 11 shows their prediction accuracy; and Figure 12 shows their training time. All models become more accurate when more queries are used for training. While Isomer is the most accurate, it uses more buckets and is much slower than others: it could not finish training in 30 minutes with 500 training queries, so the figures do not show Isomer for larger training workloads.[5] On the other hand, QuadHist, PtsHist, and QuickSel are comparable on all fronts and work very

[4]Note that this setup unfairly disadvantages PtsHist, because it uses much less space per bucket than QuickSel and QuadHist. For example, assuming 2D, each bucket (point) in PtsHist requires 2 numbers, while each bucket (2D range) in QuickSel and QuadHist requires 4, twice that of PtsHist.

[5]To be fair, we note that Isomer was not originally designed for batch training; instead, it builds its histogram by partitioning existing buckets upon observing the selectivity of each new query, which contributes to its slow training time in this setting.

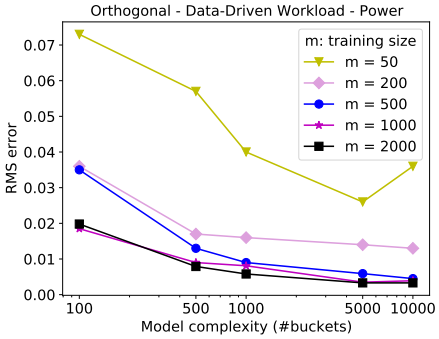

Figure 9: RMS error vs. model complexity on Data-driven workload of Power.

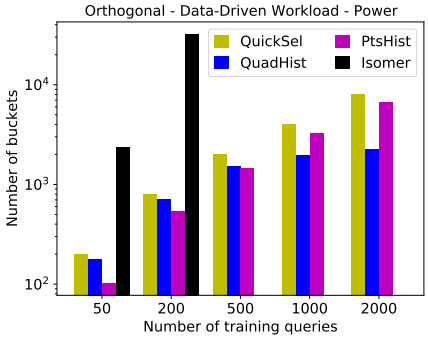

Figure 10: Model complexity on Data-driven workload of Power.

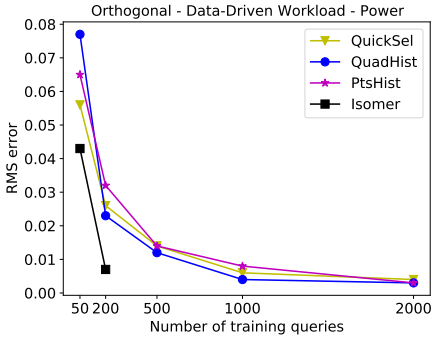

Figure 11: RMS Error vs. training size on Data-driven workload of Power.

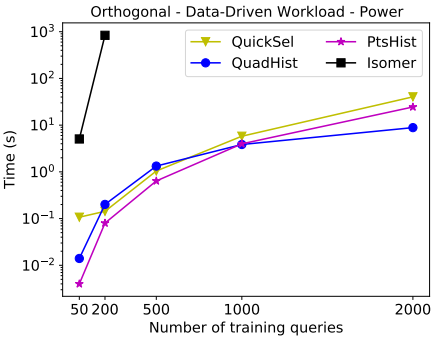

Figure 12: Training time vs. training size on Data-driven workload of Power.

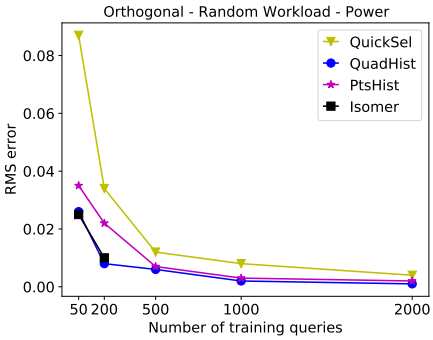

Figure 13: RMS error vs. training size on random workload of Power.

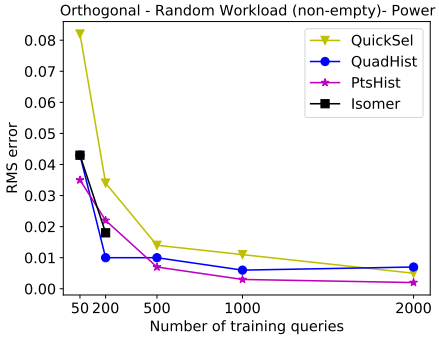

Figure 14: RMS error vs. training size on non-empty queries in random workload of Power.

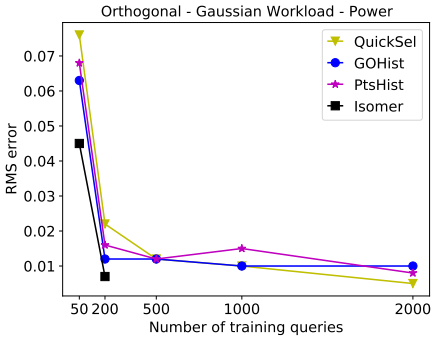

Figure 15: RMS Error vs. training size on Gaussian workload of Power.

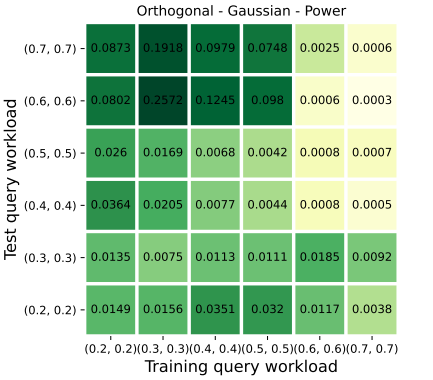

Figure 16: RMS Error vs. Difference between training and test query workload.

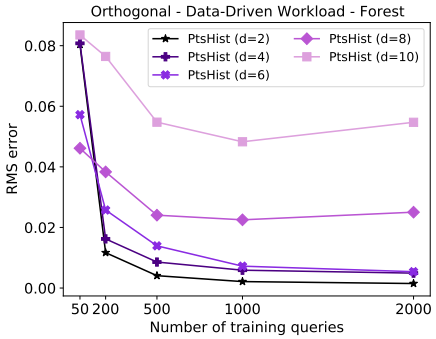

Figure 17: RMS error vs. training size on orthogonal ranges.

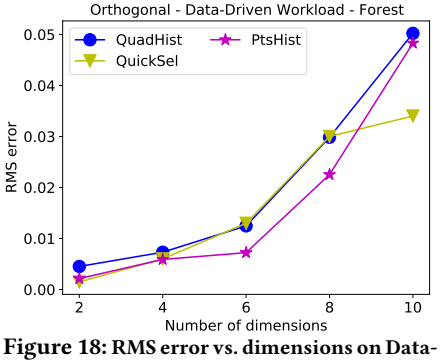

Figure 18: RMS error vs. dimensions on Data-driven workload of Forest.

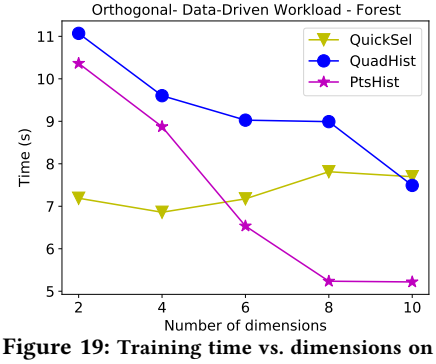

Figure 19: Training time vs. dimensions on Data-driven workload of Forest.

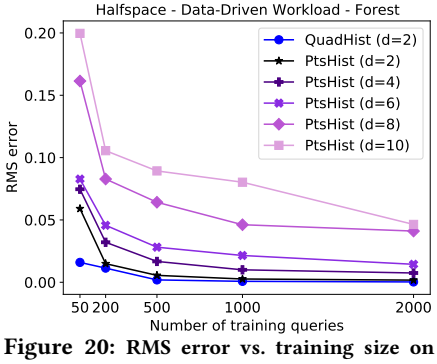

Figure 20: RMS error vs. training size on halfspace queries.

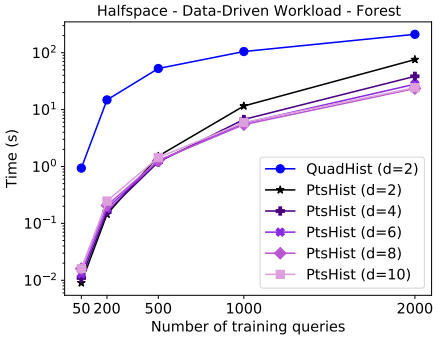

Figure 21: Training time vs. training size on halfspace queries.

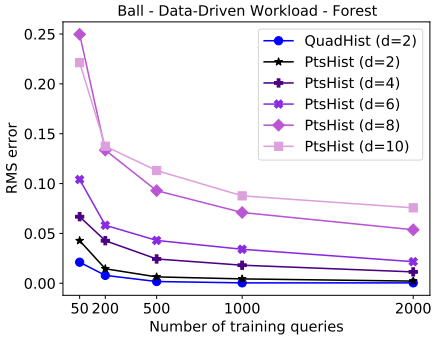

Figure 22: RMS error vs. training size on ball queries.

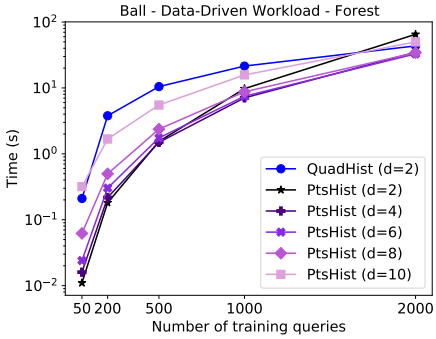

Figure 23: Training time vs. training size on ball queries.

well: their training scales to large training sets, and they are able to deliver RMS error lower than 0.01 with 1000 training queries. An encouraging observation is that despite the simple, generic nature of QuadHist and PtsHist, they are able to match (if not beat) the performance of the state-of-the-art methods. Such competitiveness can only be attributed to the fundamental learnability of the task.

Finally, another important practical consideration is prediction time. Note that all these models work similarly in terms of prediction. QuickSel, QuadHist, and Isomer have the same estimation procedure that involves computing intersections between orthogonal ranges, while PtsHist involves checking whether an orthogonal range contains a point. Therefore, their prediction time is dictated by their model complexity and already reflected in Figure 10; we do not separately report the prediction time for these different models.

4.2 Data vs. Query Distributions

So far, we have only presented results on Power data. We have also repeated these experiments in the last section on other datasets and made similar observations. Due to space constraints, we present our complete results in the Appendix B.

A second question is how the query distribution—in connection with the underlying data distribution—may affect accuracy. We note that one strength of our theoretical results in Section 2 is that they hold for arbitrary data distribution and arbitrary query distribution—even if they are highly skewed. Indeed, most real-world datasets exhibit skewness, including for example Power, and we have seen that our approach indeed works well on them. However, the query workload used so far was Data-driven. A valid question arises: what if the query workload is not correlated with the underlying data distribution? According to our theory, we should still do fine, but let us validate it empirically with other query distributions Random and Gaussian, which are independent from the underlying data distribution. Figures 32 and 35 show the prediction errors of different models, using the same exact setup on Power data as Figure 11 earlier, except we replace the Data-driven query workload with Random and Gaussian, respectively. We also report the prediction errors of non-empty queries in Random workload in Figure 14, since we have observed up to 97% Random queries with selectivity near 0. The result in Figure 14 is very similar to Figure 32, except with slight increase in the RMS error of Isomer. Overall, we can make similar observations as before—consistent with our theory, selectivity is still learnable even if query distribution is independent of data. (Figures on model complexity and training time are in the Appendix B.)

Besides RMS error, we also report Q-error results in Table 1 for all query workloads. The additional insight provided by Q-error is useful because a data-independent query workload (such as Random) on skewed data (such as Power) may result in many low-selectivity queries, where Q-error would be more informative. For example, we have observed up to 97% Random queries with selectivity near 0. From Table 1, we see that QuadHist and PtsHist, despite (and perhaps thanks to) their simplicity, are able to provide robust accuracy in terms of Q-error, often beating the more sophisticated QuickSel: QuickSel sometimes see Q-error larger than 50 even when the training size is up to 2000, while QuadHist and PtsHist have lower Q-errors even with just 50 training queries. The result on Q-error for non-empty queries are reported separately, for which PtsHist performs the best over all training sizes.

It is also instructive to dig deeper to see how QuadHist and PtsHist are able to work well on query workloads that do not correspond to the underlying data distribution. Figure 7 shows the set of buckets in QuadHist and PtsHist that are learned from the Random query workload over Power. Unknown to the learner, the real data is concentrated in the lower half. However, the Random query workload contains enough number of large queries that span both dense and sparse regions of the data space; the learner unfortunately only gets the overall selectivity of each such query, so some of the density "bleeds" into the sparse upper region, and we can see that the buckets are actually not ideal. Luckily, the subsequent weight assignment step (Section 3) mitigates this issue by assigning low weights to the upper region, making the resulting distribution more consistent with the underlying data distribution.

4.3 Training vs. Testing Query Distributions

Our next question is: what if training and testing query distributions do not match? The learning theory will not provide us with any guarantee on the performance over a different query workload, but in practice, if the test query workload is not completely disjoint from the training query workload, we should still expect to gain something from a learned model. In this set of experiments, we explore different combinations of training and testing query workloads. We

	Training	Isomer				QuickSel				QuadHist				PtsHist			
	Size	50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX
Data-driven	50	1.032	1.33	**2.046**	2.05	1.11	1.641	3.962	4.682	1.013	1.647	3.315	3.759	1.042	2.47	3.644	3.826
	200	1.006	1.051	**1.232**	1.45	1.027	1.411	1.743	2.699	1.008	1.265	1.588	1.59	1.011	1.512	1.75	2.621
	500	-	-	-	-	1.008	1.157	1.546	1.644	1.004	1.121	**1.344**	1.906	1.006	1.227	1.997	3.26
	1000	-	-	-	-	1.004	1.068	1.566	5.329	1.001	1.066	**1.097**	1.179	1.004	1.126	1.28	1.297
	2000	-	-	-	-	1.003	1.052	1.212	1.469	1.001	1.039	**1.096**	1.115	1.001	1.052	1.292	1.298
Random	50	1.149	9.819	644.864	861.236	1.154	28.949	3572.99	18401.38	1.176	8.917	31.271	35.798	1.029	5.238	**12.491**	13.154
	200	1.05	3.444	21.317	385.339	1.047	9.056	85.699	1434.633	1.015	1.972	**7.378**	9.97	1.047	7.801	24.433	46.537
	500	-	-	-	-	1.036	49.032	354.379	1277.717	1.014	1.713	16.876	20.619	1.031	2.195	**8.329**	17.208
	1000	-	-	-	-	1.025	22.38	284.91	686.495	1.006	1.829	**2.712**	9.605	1.025	3.208	5.419	13.947
	2000	-	-	-	-	1.012	5.077	32.203	53.021	1.004	1.764	**2.293**	4.439	1.006	1.731	9.95	20.35
Random (non-empty)	50	1.246	10.921	36.016	60.037	1.324	30.459	48.154	203.101	1.233	7.419	85.900	550.55	1.157	3.147	**5.018**	5.403
	200	1.053	2.047	11.491	26.253	1.11	7.091	28.577	32.144	1.050	2.731	6.140	9.720	1.111	2.797	**4.734**	10.011
	500	-	-	-	-	1.027	6.813	36.431	54.352	1.032	1.424	2.216	3.129	1.028	1.770	**2.139**	2.165
	1000	-	-	-	-	1.041	3.048	6.625	33.086	1.025	2.057	3.110	3.218	1.023	1.788	**2.540**	2.607
	2000	-	-	-	-	1.024	3.876	5.748	8.448	1.028	1.425	2.126	2.457	1.013	1.301	**1.379**	1.672
Gaussian	50	1.041	2.324	7.011	21.751	1.135	7.192	21.298	101.17	1.044	2.328	**4.954**	5.079	1.120	4.492	7.683	7.920
	200	1.009	1.146	**2.591**	3.879	1.058	13.036	36.542	152.506	1.019	2.049	3.029	4.522	1.044	2.507	5.388	6.788
	500	-	-	-	-	1.038	6.956	58.594	596.505	1.015	1.598	3.166	3.478	1.025	1.391	**1.982**	2.625
	1000	-	-	-	-	1.032	3.526	9.092	447.129	1.012	1.439	2.719	4.070	1.034	2.009	**2.457**	2.895
	2000	-	-	-	-	1.018	2.74	10.127	218.331	1.009	1.365	1.785	2.163	1.014	1.295	**1.666**	2.513

Table 1: Q-error over Power. The bold numbers are those ranking the smallest in 99th Q-error.

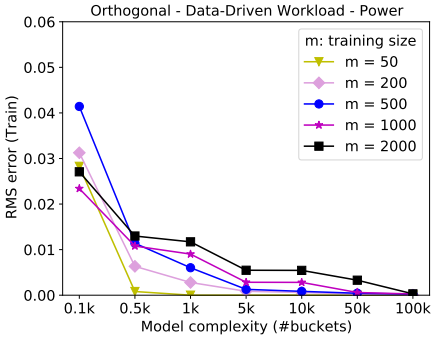

Figure 24: Train RMS error vs. RMS-Model complexity.

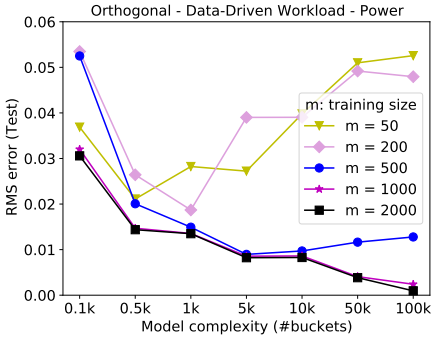

Figure 25: Test RMS error vs. RMS-Model complexity.

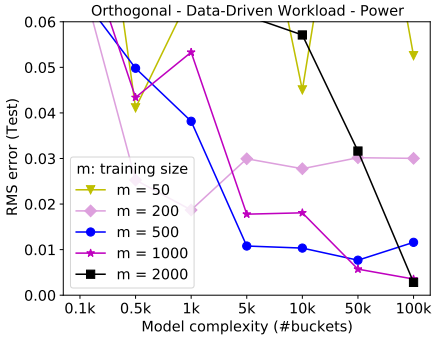

Figure 26: Test RMS error vs. L_∞-model complexity.

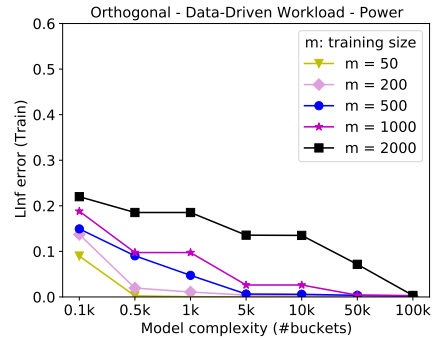

Figure 27: Train L_∞ error vs. L_∞-Model complexity

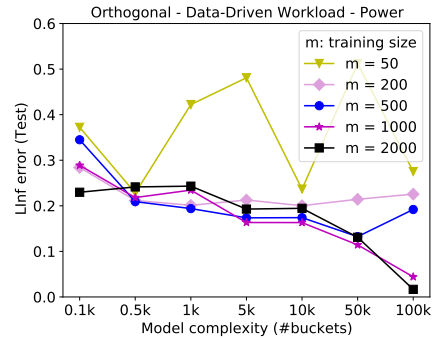

Figure 28: Test L_∞ error vs. L_∞-Model complexity.

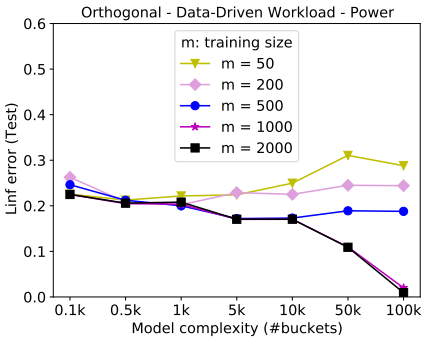

Figure 29: Test L_∞ error vs. RMS-Model complexity.

use 2D Gaussian query workloads , but shift the Gaussian distribution (from which the center points of queries are drawn) such that its mean is located at $(0.2, 0.2)$, $(0.3, 0.3)$, $(0.4, 0.4)$, $(0.5, 0.5)$, $(0.6, 0.6)$, or $(0.7, 0.7)$ (while the covariance remains at 0.033). Figure 16 shows,

as a heat map, the prediction error of QuadHist under each training/testing combination. First, we can see that when the training and test query workloads are the same (along the diagonal), the error is the smallest in most cases. If we fix the training query workload, say the column indexed by $(0.6, 0.6)$, we observe that error gradually increases with the test query workload shifting away from $(0.6, 0.6)$ to $(0.2, 0.2)$ or $(0.7, 0.7)$. Symmetrically, if we fix the test query workload, say the row indexed by $(0.7, 0.7)$, we observe that the error gradually decreases with training query workload shifting from $(0.3, 0.3)$ to $(0.7, 0.7)$. In this case, even when the shift between two query workloads is large, there is still considerable overlap between their coverage of the underlying data space; hence the error remains manageable.

4.4 Effect of Dimensionality

Recall that Theorem 2.1 shows a training size $m = \tilde{O}_d((\frac{1}{\epsilon})^{f(d)})$, where $f(d) = 2d + 3$ for orthogonal range queries, $f(d) = d + 4$ for linear inequality queries, and $f(d) = d + 5$ for distance-based queries. For all these range types, we see an exponential dependency of sampling complexity m on the dimensionality d. We now investigate this dependency experimentally. Recall that PtsHist is our method of choice in higher dimensions. For each setting of dimensionality d, we use a d-dimensional subspace of Forest and a d-dimensional Data-driven orthogonal range query workload; Figure 17 shows the error of PtsHist under different training sizes for the given d as a line plot. The model complexity of PtsHist is always set to 4× the number of training queries, consistent with earlier experiments. For each line plot, we see that the error gradually decreases with more training queries, and eventually flattens out. As we increase d, we see the series of line plots pushing away from origin. Moreover, if we set the desired accuracy by drawing a horizontal line in Figure 17, its intersections with various line plots will show that as the dimensionality goes up, the number of training queries required to achieve this accuracy also goes up, as our theory predicates. (Figures on model complexity and training time are in the Appendix B.) Since PtsHist's training time primarily depends on the model complexity, which is pegged to the training size here, we do not see significant differences among different d. However, combining this observation with the observation from Figure 17 that a higher d demands more training queries, we still conclude higher dimensions require longer training time.

Next, we compare PtsHist with other methods. Isomer is difficult to scale to higher dimensions due to the exponential dependency of its model complexity on d, so we compare with QuickSel and our own QuadHist here. As before, the number of buckets used by QuadHist and PtsHist is set to be no larger than that of QuickSel. We vary the dimensionality of Forest from 2 to 10, using 1000 Data-driven training queries in each case. Figures 18 and 19 show the error and training time of the three methods. Overall, the three methods have competitive prediction accuracy, and all see larger error in higher dimensions. Since the model complexity is fixed (across d), differences in training time primarily come from solver speed (which can be highly situational and hard to interpret) and per-bucket computational cost. Thanks to PtsHist's simpler buckets and lower per-bucket computational cost, we see that PtsHist holds significant advantage in terms of training time in high dimensions, which is the case that it is intended for.

4.5 Other Query Types

Beyond orthogonal range queries, we are also interested in other classes of range queries, such as halfspace and ball, which have many applications in databases but have seen much less work on selectivity estimation than orthogonal range queries, perhaps since the problem is perceived to be more difficult. From our theory in Section 2.2, however, selectivity functions for these queries are also learnable. This last set of experiments is designed to verify this claim.s We focus on QuadHist and PtsHist show results across different dimensions for Forest data: Figures 20, 21 for halfspace queries and Figures 22, 23 for ball queries. All query workloads are Data-driven, and the model complexity is always no more than 4× the number of training queries. We only show the results on QuadHist for $d = 2$, since in higher dimensions, its prediction for halfspace and ball queries involve complicated intersection operations that make it too slow compared with PtsHist. In Figures 20 and 22, we see that error generally decreases as we increase the training size, and higher dimensionality generally requires a bigger training size to achieve the same level of accuracy. QuadHist is more accurate than PtsHist in 2D, although it is not applicable in higher dimensions. In terms of training time shown in Figures 21 and 23, we observe that QuadHist is slow than PtsHist in 2D, and as dimensionality increases, PtsHist's training procedure remains very scalable, because its complexity is primarily dependent on its model complexity instead of d. Overall, we observe that as simple as it is, PtsHist provides a reasonable solution for learning the selectivity of halfspace and ball queries, thanks to the theoretical guarantees on the learnability of these query classes.

4.6 Objective Functions: L_2 vs. L_∞

As our theoretical framework can be adapted to other error metric, such as L_2 error and L_∞ error, we also perform experiments to test the objective function on the learned model. We first use the L_2 error as the objective function to train the model, and obtain the RMS error over training queries in Figure 24, the RMS error over test queries in Figure 25 and the L_∞ error over test queries in Figure 29. In addition, we also use the L_∞ error as the objective function to train the model, and obtain the L_∞ error over training queries in Figure 27, the L_∞ error over test queries in Figure 28 and the RMS error over test queries in Figure 26. A common observation is that when the model is trained by L_2 (resp. L_∞) error, the train L_2 (resp. L_∞) error is always smaller than the test L_2 (resp. L_∞) error. Meanwhile, the model trained by L_2 error can also have good predication in terms of L_∞ error (see Figure 29), while the model trained by L_∞ error does not have any guarantee on the predication in terms of L_2 error (see Figure 26). Overall, we observe that L_2 norm is a better objective function than L_∞ on the Power data.

Summary. Our main findings can be summarised as follow:

- The empirical results have verified our learning theory by showing a clear dependency between error ϵ, training size m and dimension d. When fixing d, more training samples lead to more accurate modeling of data distribution, thus smaller predication

error. On the other hand, when fixing m, higher dimension leads to more coarse-grained modeling of data distribution, thus larger prediction error. (Section 4.1 and 4.4)

- The empirical results have verified that our learning theory holds for arbitrary data distribution and query distribution, even if they are highly skewed. (Section 4.2)
- Our learning theory does not provide us with any guarantee on the performance of learned model when the training and test query distributions do not match. The empirical results imply that we will get the most accurate model when training and test query distribution matches exactly, but we can still gain something from a learned model when there is overlap between their coverage of the underlying data space. (Section 4.3)
- Using different objective functions will lead to learned models with different predication ability. In our empirical study, the L_2 norm is a better candidate for training the model than the L_∞ norm. (Section 4.6)
- Two simple algorithms proposed in Section 3 work very well in practice, matching or even outperforming the state-of-art methods. QuadHist stands out for efficiency and accuracy in lower dimensional space while PtsHist scales better in higher dimensional space. (Section 4.1, 4.2 and 4.4)

5 RELATED WORK

Orthogonal range queries. There has been much work on learnability of the selectivity estimation for orthogonal range queries. We summarize them in Table 2 (see [46] for a comprehensive review) under two metrics: methodology and input. First, there are two main approaches in tackling this problem: one is to build a mapping between queries and their selectivities via feature vectors, and the other is to learn the underlying data distribution. Secondly, depending on the input of learned models, we divide them into three cases: only the underlying data, only the previous queries from workload, and both data and query (hybrid). Combing these two metrics, we review the literature from the following four categories:

- **Regression model with hybrid input.** MSCN [22] represents a query as a feature vector which contains three modules (i.e., table, join, and predicate) and uses a *multi-set convolutional network* for training. MSCN enriches the training queries with materialized samples from underlying data. LW [13] a lightweight selectivity estimation method, uses both query range and heuristic selectivity estimators as features. It adopts both neural network and gradient boost tree model separately for training.
- **Regression model only with query as input.** DQM [15] proposes one-hot encoding to encode categorical attributes (and treats numerical attributes as categorical attributes by automatic discretization), and uses a neural network for training.
- **Data distribution model only with data as input.** This line of work takes samples from the underlying data distribution. Both Naru [49] and DQM-D [28] decompose the data distribution into conditional data distributions using the product rule: Naru uses *progressive sampling* to sample values attribute by attribute according to the internal output of conditional probability distribution, and DQM-D selects samples in proportional to the contribution they make to the query cardinality according to the result from the previous stage. DeepDB [18] builds *sum-product networks* on random data samples to capture the data distribution.
- **Data distribution model only with query as input.** As far as we are aware, query-driven histograms [10, 19, 27, 28, 36, 39] are the methods that learn data distribution only from previous queries, for example STHoles [10] exploits results of queries in the workload and gathers associated statistics to progressively build and refine a histogram; Isomer [39] applies STHoles for histogram bucket creations, and computes the density for buckets by maximizing entropy distribution; QuickSel [36] uses a mixture model of uniform distributions to represent the underlying data distribution, which can be viewed as overlapping histograms.

Models	Data	Hybrid	Query
Regression	–	MSCN [22] LW [13]	DMQ-Q [15]
Data Distribution	Naru [49] DeepDB [18] DMQ-D [15]	–	STHoles [10] Isomer [39] QuickSel [36]

Table 2: Taxonomy of Learned Cardinality Estimation.

Distance-based queries. Deep learning has been studied to estimate the selectivity for distance-based queries. Wang et al. [47] learned the cardinality by resorting to the VAE (Variational Autoencoders) and embeddings for different thresholds separately for enhancing accuracy and guaranteeing monotonicity. Sun et al. [41] utilized deep neural network to learn cardinality and adopt two strategies to improve the accuracy and reduce the size of training data, i.e., query segmentation and data segmentation. There are some other methods depending on the underlying data, for example, clustering-based methods [9], kernel-based methods [30].

Learning theory. It is beyond the scope of this paper to review the relevant work on learning theory and we refer reader to a few classical books on this topic [5, 21, 44]. In the literature of ML, Valiant's pioneering work [43] first established the notion of learnability and the framework of *probably approximately correct (PAC)* learning. Intuitively, the learner in this framework receives random samples from underlying training data, and aims to select a hypothesis from a set of possible hypotheses which has low generalization error (ϵ) with high probability ($1 - \delta$) on the unobserved samples from the same distribution. Surprisingly, relationships between the PAC-learnability of a hypothesis class and its inherent properties have been proved. For example, a Boolean hypothesis class is learnable if and only if it has finite VC-dimension [5, 44, 45], and a real-valued function class is PAC-learnable if and only if it has finite fat-shattering dimension [5, 7, 20]. This framework of PAC learning is exactly the theoretical foundation behind our investigation of learnability of selectivity functions.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented an ML-based technique for estimating the selectivity of selection queries in DB systems. Central to our approach were generalization bounds that we proved for this problem (Theorem 2.1), thereby establishing a formal framework for applying classical ML theory (PAC learning) to DB problems. In contrast, the predominant approach in previous work has been to use deep

learning techniques, which have consistently outperformed traditional optimization methods for a range of important problems in DB research. However, in spite of much empirical success, obtaining generalization bounds for deep learning remains one of the outstanding open challenges of the modern era. We expect the trend of using ML for DB will accelerate even further in the future, and hope that our work in initiating a formal study of the *learnability* of DB problems will complement the existing efforts at leveraging deep learning for improving the performance of DB systems in practice. There are several interesting directions for future work. As mentioned earlier, understanding the complexity of finding an optimal distribution with a given model complexity is an open problem. Although our framework does not assume query ranges to be bounded and thus works even if we consider data distributions with unbounded support, e.g., Gaussian mixtures, developing an algorithm that computes a Gaussian mixture (or another model) with a small loss given a training sample is also an open problem. Developing theory and algorithms for learning selectivity of range queries of practical interest with unbounded VC dimensions and extending the learning framework to unsupervised approaches are other intriguing directions for future work.

REFERENCES

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.nnls.html.
[2] https://github.com/huxiao2010/Selectivity.
[3] P. K. Agarwal and M. Sharir. 2000. Arrangements and their applications. In *Handbook of computational geometry*. Elsevier, 49–119.
[4] N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. 1997. Scale-sensitive dimensions, uniform convergence, and learnability. *JACM* 44, 4 (1997), 615–631.
[5] M. Anthony and P. L. Bartlett. 2009. *Neural network learning: Theoretical foundations*. cambridge university press.
[6] Peter L Bartlett and Philip M Long. 1995. More theorems about scale-sensitive dimensions and learning. In *Proceedings of the eighth annual conference on Computational learning theory*. 392–401.
[7] P. L. Bartlett, P. M. Long, and R. C. Williamson. 1996. Fat-shattering and the learnability of real-valued functions. *J. Comput. Syst. Sci.* 52, 3 (1996), 434–452.
[8] S. Ben-David and M. Lindenbaum. 1998. Localization vs. identification of semi-algebraic sets. *Machine Learning* 32, 3 (1998), 207–224.
[9] N. Brisaboa, O. Pedreira, D. Seco, R. Solar, and R. Uribe. 2008. Clustering-based similarity search in metric spaces with sparse spatial centers. In *SOFSEM*. Springer, 186–197.
[10] N. Bruno, S. Chaudhuri, and L. Gravano. 2001. STHoles: A multidimensional workload-aware histogram. In *Proc. 20th ACM SIGMOD Int. Conf. Management Data*. 211–222.
[11] B. Chazelle and E. Welzl. 1989. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete & Computational Geometry* 4, 5 (1989), 467–489.
[12] D. Dua and C. Graf. 2017. UCI machine learning repository. (2017). http://archive.ics.uci.edu/ml/index.php
[13] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri. 2019. Selectivity estimation for range predicates using lightweight models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057.
[14] W. R. Gilks, S. Richardson, and D. Spiegelhalter. 1995. *Markov chain Monte Carlo in practice*. CRC press.
[15] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *Proc. 39th ACM SIGMOD Int. Conf. Management Data*. 1035–1050.
[16] D. Haussler. 1992. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.* 100, 1 (1992), 78–150.
[17] D. Haussler and E. Welzl. 1987. ϵ-nets and simplex range queries. *Discret. Comput. Geom.* 2, 2 (1987), 127–151.
[18] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. 2019. DeepDB: learn from data, not from queries! *Proc. VLDB Endow.* 13, 7 (2019), 992–1005.
[19] R. Kaushik and D. Suciu. 2009. Consistent histograms in the presence of distinct value counts. *Proc. VLDB Endow.* 2, 1 (2009), 850–861.
[20] M. J. Kearns and R. E. Schapire. 1994. Efficient distribution-free learning of probabilistic concepts. *J. Comput. System Sci.* 48, 3 (1994), 464–497.
[21] M. J. Kearns and U. Vazirani. 1994. *An introduction to computational learning theory*. MIT press.
[22] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper. 2019. Learned cardinalities: Estimating correlated joins with deep learning. (2019).
[23] R. J. Lipton, J. F. Naughton, and D. A. Schneider. 1990. Practical selectivity estimation through adaptive sampling. In *Proc. 9th ACM SIGMOD Int. Conf. Management Data*. 1–11.
[24] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska. 2020. Bao: Learning to Steer Query Optimizers. *arXiv preprint arXiv:2004.03814* (2020).
[25] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. 2019. Neo: A learned query optimizer. 12, 11 (2019), 1705–1718.
[26] R. Marcus and O. Papaemmanouil. 2018. Deep reinforcement learning for join order enumeration. In *aiDM*. 1–4.
[27] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. 2007. Consistent selectivity estimation via maximum entropy. *The VLDB journal* 16, 1 (2007), 55–76.
[28] V. Markl, N. Megiddo, M. Kutsch, T. M. Tran, P. Haas, and U. Srivastava. 2005. Consistently estimating the selectivity of conjuncts of predicates. In *Proc. 31th Very Large Data Bases*. 373–384.
[29] Y. Matias, J. S. Vitter, and M. Wang. 1998. Wavelet-based histograms for selectivity estimation. In *Proc. 17th ACM SIGMOD Int. Conf. Management Data*. 448–459.
[30] M. Mattig, T. Fober, C. Beilschmidt, and B. Seeger. 2018. Kernel-Based Cardinality Estimation on Metric Data. In *EDBT*. 349–360.
[31] G. Moerkotte, T. Neumann, and G. Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proc. VLDB Endow.* 2, 1 (2009), 982–993.
[32] S. Muthukrishnan, V. Poosala, and T. Suel. 1999. On rectangular partitionings in two dimensions: Algorithms, complexity and applications. In *ICDT*. Springer, 236–256.
[33] P. Negi, R. Marcus, H. Mao, N. Tatbul, T. Kraska, and M. Alizadeh. 2020. Cost-Guided Cardinality Estimation: Focus Where it Matters. In *Proc. 36th Annu. IEEE Int. Conf. Data Eng.* IEEE, 154–157.
[34] S. of New York. 2019. Vehicle, snowmobile, and boat registrations. (2019). catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registration
[35] J. Pach and P. K. Agarwal. 2011. *Combinatorial geometry*. Vol. 37. John Wiley & Sons.
[36] Y. Park, S. Zhong, and B. Mozafari. 2020. Quicksel: Quick selectivity learning with mixture models. In *Proc. 39th ACM SIGMOD Int. Conf. Management Data,*. 1017–1033.
[37] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. 1996. Improved histograms for selectivity estimation of range predicates. *ACM Sigmod Record* 25, 2 (1996), 294–305.
[38] V. Poosala and Y. E. Ioannidis. 1997. Selectivity estimation without the attribute value independence assumption. In *VLDB*, Vol. 97. Citeseer, 486–495.
[39] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. 2006. Isomer: Consistent histogram construction using query feedback. In *Proc. 22th Annu. IEEE Int. Conf. Data Eng.* 39–39.
[40] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. 2008. SPARQL basic graph pattern optimization using selectivity estimation. In *Proc. 17th Int. Conf. World Wide Web*. 595–604.
[41] J. Sun, G. Li, and N. Tang. 2021. Learned Cardinality Estimation for Similarity Queries. In *Proc. 40th ACM SIGMOD Int. Conf. Management Data*.
[42] G. T. Toussaint. 1983. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, Vol. 83. A10.
[43] L. G. Valiant. 1984. A theory of the learnable. *Commun. ACM* 27, 11 (1984), 1134–1142.
[44] V. Vapnik. 2013. *The nature of statistical learning theory*. Springer science & business media.
[45] V. N. Vapnik and A. Y. Chervonenkis. 2015. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*. Springer, 11–30.
[46] X. Wang, C. Qu, W. Wu, J. Wang, and Q. Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proc. VLDB Endow.* 14, 9 (2021), 1640–1654.
[47] Y. Wang, C. Xiao, J. Qin, X. Cao, Y. Sun, W. Wang, and M. Onizuka. 2020. Monotonic cardinality estimation of similarity selection: A deep learning approach. In *Proc. 39th ACM SIGMOD Int. Conf. Management Data*. 1197–1212.
[48] R. S. Wenocur and R. M. Dudley. 1981. Some special vapnik-chervonenkis classes. *Discrete Mathematics* 33, 3 (1981), 313–318.
[49] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. 2019. Deep unsupervised cardinality estimation. *Proc. VLDB Endow.* 13, 3 (2019), 279–292.

A SUPPLEMENTARY MATERIALS IN SECTION 3

A.1 Analysis of QuadHist

To illustrate the high-level idea, we give the pseudocode for QuadHist in 2D below. Let $\mathbf{R}$ be the set of training samples.

Algorithm 1: QuadHist($\mathbf{R}, \tau$)

```
1 r ← [0,1] × [0,1];
2 T ← {r};
3 foreach R ∈ R do
4 |   T ← UpdateQuad(T, r, R, τ);
5 Return T;
```

Algorithm 2: UpdateQuad($\mathcal{T}, u, R, \tau$)

```
1 p ← (|u∩R| / |R|) · s(R);
2 if p > τ then
3 |   if u is a leaf node then
4 |   |   split u into four equal-sized nodes u1, u2, u3, u4;
5 |   foreach node v as the child of u do
6 |   |   T ← UpdateQuad(T, v, R, τ);
7 return T;
```

The input of Algorithm 1 is $\mathbf{R}$ as a set of training queries in terms of $\langle R, s(R)\rangle$ pairs, and a parameter $0 < \tau < 1$. We start with an root node corresponding to the whole region $[0,1] \times [0,1]$. Then, we update the quadtree $\mathcal{T}$ incrementally with training queries one by one. The detailed procedure for the update with one training query is described UpdateQuad.

The input of Algorithm 2 is a quadtree $\mathcal{T}$, one node $u \in \mathcal{T}$, a query R, and parameter $0 < \tau < 1$. The algorithm updates node u using R (i.e., a rectangle in this scenario), and distinguishes two cases. If u is a leaf node, we only need to split u when the selectivity of intersection area $u \cap R$ exceeds the threshold τ (line 2), assuming the distribution inside R is uniform. In that case, we further invoke this procedure for each children of u recursively (line 5-6). Otherwise, u is an internal node, and we also invoke this procedure for each children of u recursively. An example is given in Figure 7.

Analysis. We first show an interesting theoretical property of Algorithm 1 in Lemma A.4. Intuitively, the partition is oblivious to the ordering of input queries. Hence, the space usage in terms of number of buckets only depends on the training set. Moreover, this model is always deterministic, i.e., the model always gives the same predication results as long as it is trained by the same set of queries.

Lemma A.1. *Algorithm 1 generates the same partition, regardless of the ordering of inserting training queries.*

Proof of Lemma A.4. Consider an arbitrary ordering of training queries $\mathbf{R}$. Let $\mathcal{T}$ be the resulted quadtree. From Algorithm 2, we first observe that (1) for any leaf node $u \in \mathcal{T}$, $\frac{|u\cap R|}{|R|} \cdot s(R) \le \tau$ holds fore every query $R \in \mathbf{R}$; and (2) for any non-leaf node $u \in \mathcal{T}$, there must exist a query $R \in \mathbf{R}$ such that $\frac{|u\cap R|}{|R|} \cdot s(R) > \tau$.

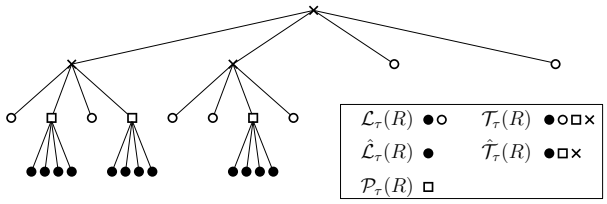

Figure 30: An illustration of nodes visited by R in Figure 6. We also illustrate $\mathcal{T}_\tau(R)$, $\hat{\mathcal{T}}_\tau(R)$, $\mathcal{L}_\tau(R)$, $\hat{\mathcal{L}}_\tau(R)$ and $\mathcal{P}_\tau(R)$ used by the proof of Lemma A.2.

Consider another ordering of training queries $\mathbf{R}$, and the resulted quadtree $\mathcal{T}'$. It suffices to show that $\mathcal{T}$ and $\mathcal{T}'$ have the same set of leaf nodes. Suppose not, assume u is a leaf node in $\mathcal{T}$ but not a leaf in $\mathcal{T}'$. From the construction of $\mathcal{T}$, the following must hold:

- there must exist a query $R \in \mathbf{R}$ such that $\frac{|p(u)\cap R|}{|R|} \cdot s(R) > \tau$, where $p(u)$ is the parent of u in $\mathcal{T}$; and
- $\frac{|u\cap R|}{|R|} \cdot s(R) \le \tau$ holds for every query $R \in \mathbf{R}$.

Hence, there are two possible cases of u in $\mathcal{T}'$: (i) u appears in $\mathcal{T}'$, i.e., u is an internal node; or (ii) u does not appear in $\mathcal{T}'$, then there exists an ancestor u' of u such that u' is a leaf node in $\mathcal{T}'$. Implied by the construction of $\mathcal{T}'$, we find that: for (i) there must exist a query $R \in \mathbf{R}$ such that $\frac{|u\cap R|}{|R|} \cdot s(R) > \tau$, coming to a contradiction; and for (ii), there exists a query $R \in \mathbf{R}$ such that $\frac{|u'\cap R|}{|R|} \cdot s(R) < \tau$, coming to a contradiction that $|p(u) \cap R| \le |u' \cap R|$ since u' is also an ancestor of $p(u)$ or $u' = p(u)$. In this way, every leaf node in $\mathcal{T}$ is also a leaf node in $\mathcal{T}'$. Applying the similar argument, we can show every leaf node in $\mathcal{T}'$ is also a leaf node in $\mathcal{T}$. Thus, $\mathcal{T}$ and $\mathcal{T}'$ have the same set of leaf nodes, i.e., the same partition. □

However, we don't have a good theoretical bound on the number of leaf nodes, i.e., the number of buckets in the histogram, which may depend on the data distribution, query distribution, training queries, threshold of τ, etc. We provide some statistics on this quantity in the practical evaluation part (Section 4), but at least we can manually set a limit on the maximum number of leaf nodes in the construction. We next study the time complexity of Algorithm 1, which turns out to only depend on the training set. More specifically, we show that the run-time of each invocation of UpdateQuad (line 4) in terms of the number of nodes visited by R.

Lemma A.2. *For an arbitrary query R and parameter τ, the number of nodes visited by UpdateQuad$(\mathcal{T}, r, R, \tau)$ is $O\left(\frac{s(R)}{\tau} \cdot \log_2 \frac{s(R)}{\tau \cdot |R|}\right)$.*

Proof of Lemma A.2. We give an example in Figure 30 to illustrate the notations to be used in this proof.

Let $\mathcal{T}_\tau(R)$ be the set of nodes visited by query R. Let $\mathcal{L}_\tau(R)$ be the set of leaves in the quatree returned by GeometryOblivious$(\{R\}, \tau)$. Careful inspection reveals that $\mathcal{T}_\tau(R)$ is exactly the set of nodes in $\mathcal{L}_\tau(R)$ together with their ancestors, i.e.,

$$\mathcal{T}_\tau(R) = \mathcal{L}_\tau(R) \cup \{u' \text{ is an ancestor of } u : u \in \mathcal{L}_\tau(R)\}$$

no matter when R is inserted. We start by considering a subset of nodes $\hat{\mathcal{L}}_\tau(R) \subseteq \mathcal{L}_\tau(R)$. A node $u \in \mathcal{L}_\tau(R)$ will not be included by

$\hat{\mathcal{L}}_\tau(R)$, if there is one sibling[6] node v of u that will be further split by R, i.e., v has descendant in $\mathcal{L}_\tau(R)$. Correspondingly, let $\hat{\mathcal{T}}_\tau(R)$ be the set of nodes in $\mathcal{L}_\tau(R)$ together with their ancestors, i.e.,

$$\hat{\mathcal{T}}_\tau(R) = \hat{\mathcal{L}}_\tau(R) \cup \{u' \text{ is an ancestor of } u : u \in \hat{\mathcal{L}}_\tau(R)\}$$

A critical observation is that $|\mathcal{T}_\tau(R)| \leq 4 \cdot |\hat{\mathcal{T}}_\tau(R)|$, which lays out the foundation of our proof.

We take the union of the parents of nodes in $\hat{\mathcal{L}}_\tau(R)$ as

$$\mathcal{P}_\tau(R) = \{u \text{ is the parent of } u' : u' \in \hat{\mathcal{L}}_\tau(R)\}.$$

It can be proved that all nodes in $\mathcal{P}_\tau(R)$ are disjoint. Suppose not, assume there exists a pair of nodes $u, v \in \mathcal{P}_\tau(R)$ such that $u \cap v \neq \emptyset$. In the quadtree, there must be either $u \subseteq v$ or $v \subseteq u$. Without loss of generality, assume $u \subseteq v$, i.e., v is an ancestor of u. Since v is added to $\mathcal{P}_\tau(R)$, there must exist at least one child u' of v that is also included by $\hat{\mathcal{L}}_\tau(R)$. Moreover, u' does not lie on the path from v to u; otherwise, u' will be further split by R and won't be included by $\mathcal{L}_\tau(R)$. Let u'' be another child of v that lies on the path from v to u. Due to the construction of $\hat{\mathcal{L}}_\tau(R)$, u' won't be added to $\hat{\mathcal{L}}_\tau(R)$ due to u'', coming to a contradiction. Moreover, for each node $u \in \mathcal{P}_\tau(R)$, we observe that $|u \cap R| \geq |R| \cdot \tau / s(R)$ since u will be further split in Algorithm 2. As proved above, all nodes in $\mathcal{P}_\tau(R)$ are disjoint, so

$$|R| \geq \sum_{u \in \mathcal{P}_\tau(R)} |u \cap R| \geq |\mathcal{P}_\tau(R)| \cdot |R| \cdot \frac{\tau}{s(R)}.$$

In this way, we show that $|\mathcal{P}_\tau(R)| \leq \frac{s(R)}{\tau}$. Implied by the fact that $|\hat{\mathcal{L}}_\tau(R)| \leq 4 \cdot |\mathcal{P}_\tau(R)|$, we obtain $|\hat{\mathcal{L}}_\tau(R)| = O(\frac{s(R)}{\tau})$.

Let u^* be the one with smallest area in $\mathcal{P}_\tau(R)$. From the analysis above, $|u^*| \geq |u^* \cap R| \geq \frac{|R|}{s(R)} \cdot \tau$. Assume u^* has dimension of $\frac{1}{2^j} \times \frac{1}{2^j}$, where $j = \lfloor \log_4 \frac{s(R)}{\tau \cdot |R|} \rfloor$. In this way, each node in $\hat{\mathcal{L}}_\tau(R)$ has at most $j+1$ ancestors. Summing over all nodes in $\hat{\mathcal{L}}_\tau(R)$, the number of nodes in $\hat{\mathcal{T}}_\tau(R)$ can be bounded by $O\left(\frac{s(R)}{\tau} \cdot \log_2 \frac{s(R)}{\tau \cdot |R|}\right)$. □

Lemma A.3. *For a parameter $0 < \tau < 1$, Algorithm 1 has time complexity of $O\left(\frac{1}{\tau} \cdot (\sum_{R \in \mathbf{R}} s(R)) \cdot \max_{R \in \mathbf{R}} \log_2 \frac{s(R)}{\tau \cdot |R|}\right)$.*

Lemma A.4. *Algorithm 1 generates the same partition, regardless of the ordering of inserting training queries.*

Proof of Lemma A.4. Consider an arbitrary ordering of training queries $\mathbf{R}$. Let $\mathcal{T}$ be the resulted quadtree. From Algorithm 2, we first observe that (1) for any leaf node $u \in \mathcal{T}$, $\frac{|u \cap R|}{|R|} \cdot s(R) \leq \tau$ holds fore every query $R \in \mathbf{R}$; and (2) for any non-leaf node $u \in \mathcal{T}$, there must exist a query $R \in \mathbf{R}$ such that $\frac{|u \cap R|}{|R|} \cdot s(R) > \tau$.

Consider another ordering of training queries $\mathbf{R}$, and the resulted quadtree $\mathcal{T}'$. It suffices to show that $\mathcal{T}$ and $\mathcal{T}'$ have the same set of leaf nodes. Suppose not, assume u is a leaf node in $\mathcal{T}$ but not a leaf in $\mathcal{T}'$. From the construction of $\mathcal{T}$, the following must hold: (1) there must exist a query $R \in \mathbf{R}$ such that $\frac{|p(u) \cap R|}{|R|} \cdot s(R) > \tau$, where $p(u)$ is the parent of u in $\mathcal{T}$ and (2) $\frac{|u \cap R|}{|R|} \cdot s(R) \leq \tau$ holds for every query $R \in \mathbf{R}$. So, there are two possible cases of u in $\mathcal{T}'$: (i) u appears in $\mathcal{T}'$, i.e., u is an internal node; or (ii) u does not appear in $\mathcal{T}'$, then there exists an ancestor u' of u such that u' is a leaf node in $\mathcal{T}'$. Implied by the construction of $\mathcal{T}'$, we find that: for (i) there must exist a query $R \in \mathbf{R}$ such that $\frac{|u \cap R|}{|R|} \cdot s(R) > \tau$, coming to a contradiction; and for (ii), there exists a query $R \in \mathbf{R}$ such that $\frac{|u' \cap R|}{|R|} \cdot s(R) < \tau$, coming to a contradiction that $|p(u) \cap R| \leq |u' \cap R|$ since u' is also an ancestor of $p(u)$ or $u' = p(u)$. In this way, every leaf node in $\mathcal{T}$ is also a leaf node in $\mathcal{T}'$. Applying the similar argument, we can show every leaf node in $\mathcal{T}'$ is also a leaf node in $\mathcal{T}$. Thus, $\mathcal{T}$ and $\mathcal{T}'$ have the same set of leaf nodes. □

[6]Two nodes in a quadtree are siblings if they have the same parent.

A.2 Reject Sampling

Drawing a uniform sample from a rectangle can be done easily by independently drawing uniform sample in each dimension, which is an interval. However, this problem becomes very challenging for general range queries, such as halfspaces and ℓ_p balls, due to their implicit dependency among different dimensions. We adopt the common solution of *rejected sampling* to tackle this issue:

- we first draw a uniform sample from the *smallest bounding box* of the range query. Note that the smallest bounding box for a geometry shape is the (hyper-)rectangle with the smallest measure (area, volume, or hyper-volume) fully containing this shape; and
- we then reject it with probability $1 - \frac{a}{b}$, assuming a is the area of input range and b is the area of its smallest bounding box.

For an ℓ_p-ball, the smallest bounding box can be easily obtained by its center and radius in each dimension. For a halfspace, the smallest bounding box can be found by the following procedure: Consider an example of halfspace query $R = (\theta_1, \theta_2, \cdots, \theta_d, b)$. Let $\hat{R} = \times_{i=1}^{d} [l_i, r_i]$ be the bounding box of R, where $[l_i, r_i]$ be the interval in the i-th dimension. Initially, we set $l_i = 0$ and $r_i = 1$ for each $i \in [d]$. For each $i \in [d]$,

- if $\theta_i > 0$, update l_i with $\frac{1}{|\theta_i|} \cdot \left(b - \sum_{j \in [d]: j \neq i} \max\{\theta_i \cdot l_i, \theta_i \cdot r_i\}\right)$ if it is larger than l_i;
- if $\theta_i < 0$, update r_i with $\frac{1}{|\theta_i|} \cdot \left(\sum_{j \in [d]: j \neq i} \max\{\theta_i \cdot l_i, \theta_i \cdot r_i\} - b\right)$ if it is smaller than r_i.

We repeatedly apply the procedures above until there is no more change on $\hat{R}$, which is exactly the smallest bounding box of R.

B SUPPLEMENTARY MATERIALS IN SECTION 4

B.1 Power

We add the omitted results (about model complexity, RMS error, and training time) over data-driven, random and Gaussian workloads of Power dataset, in Figure 10-12, Figure 31-33, and Figure 34-36.

B.2 Forest

We add the omitted results (about model complexity, RMS error, and training time) over data-driven, random, and Gaussian workloads of Forest dataset, in Figure 37-39, Figure 40-42, and Figure 43-45. The Q-error of Forest dataset is presented in Table 3.

B.3 DMV and Census

The basic information of DMV and Census datasets are as follows:

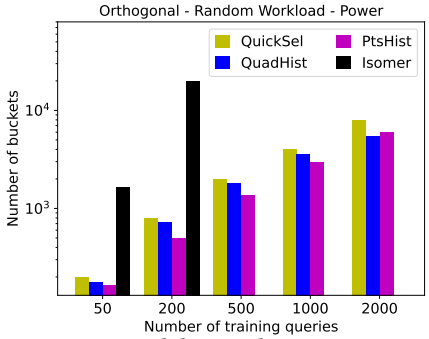

Figure 31: Model complexity over random workload of Power.

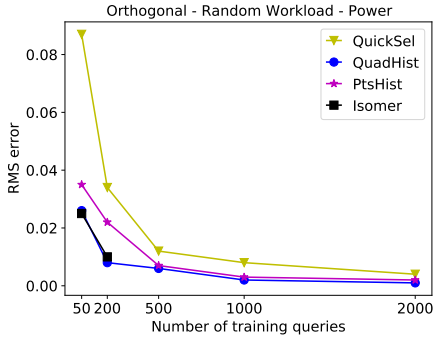

Figure 32: RMS Error vs. training size on random workload of Power.

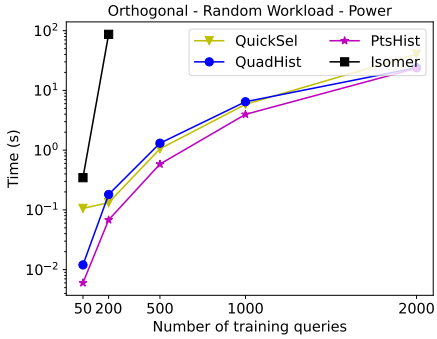

Figure 33: Training time v.s. training size over random workload of Power.

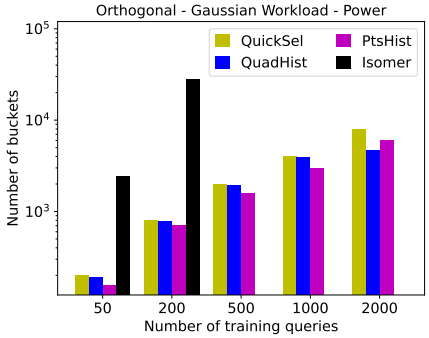

Figure 34: Model complexity over Gaussian workload of Power.

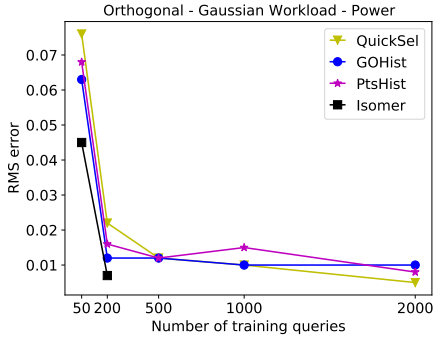

Figure 35: RMS Error vs. training size on Gaussian workload of Power.

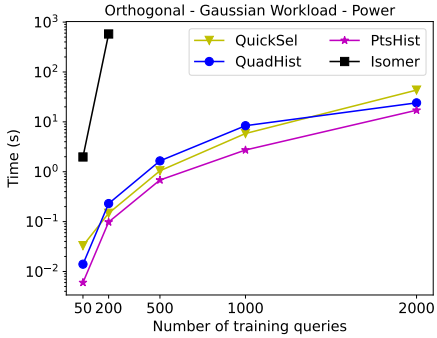

Figure 36: Training time v.s. training size over Gaussian workload of Power.

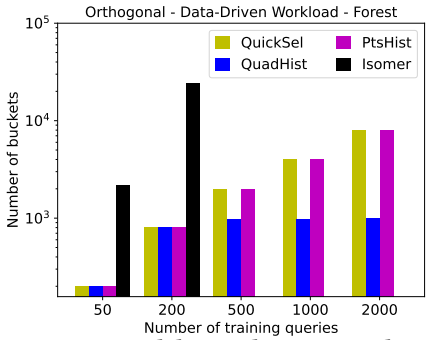

Figure 37: Model complexity over data-driven workload of Forest.

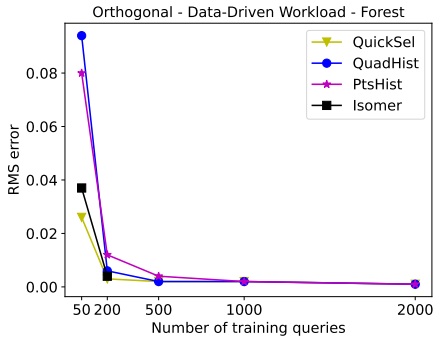

Figure 38: RMS Error vs. training size on data-driven workload of Forest.

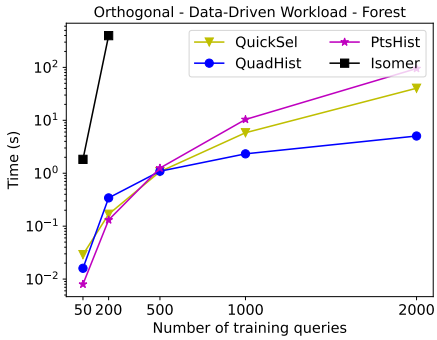

Figure 39: Training time vs. training size on data-driven workload of Forest.

- **Census [12]** contains the basic population characteristics, with 49K tuples over 13 attributes (8 categorical and 5 numerical).
- **DMV [34]** contains the vehicle registration records of New York State, with 11M tuples over 11 attributes (10 categorical and 1 numerical).

Similar to the Power dataset, we generate the query workloads over Census and DMV datasets for orthogonal queries. Firstly, we randomly pick a subset of attributes and project all the tuples on those chosen columns. If a categorical attribute is selected, we discretize the values in its domain and normalize them into [0,1]. Then we generate each query by sampling the query center and the random range, following the similar steps in Section 4. Here, we focus on data-driven workload to verify our theory.

The model complexity, RMS error, training time and Q-error of different methods over Census dataset are shown in Figure 49, Figure 50, Figure 51 and Table 5. The model complexity, RMS error, training time and Q-error of different methods over DMV dataset are shown in Figure 46, Figure 47, Figure 48 and Table 4.

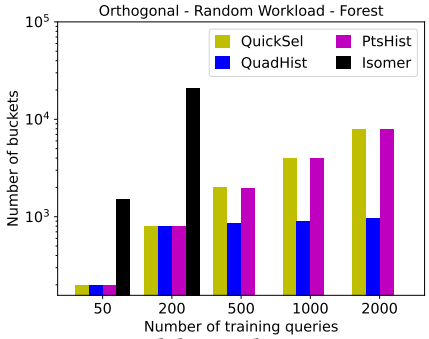

Figure 40: Model complexity over Random workload of Forest.

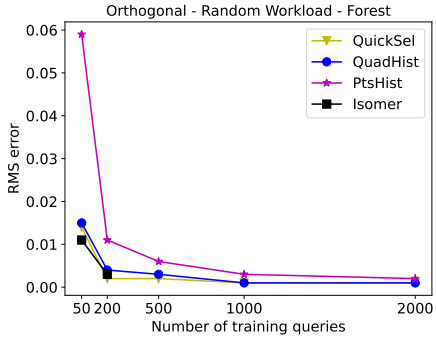

Figure 41: RMS Error vs. training size on Random workload of Forest.

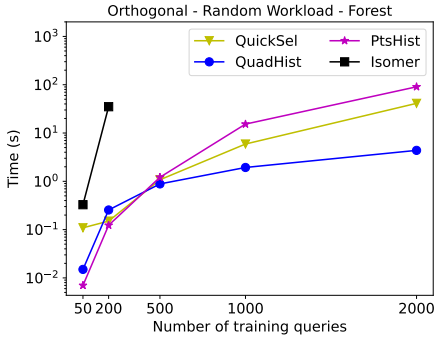

Figure 42: Training time vs. training size on Random workload of Forest.

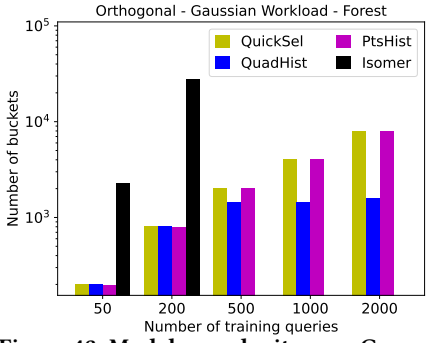

Figure 43: Model complexity over Gaussian workload of Forest.

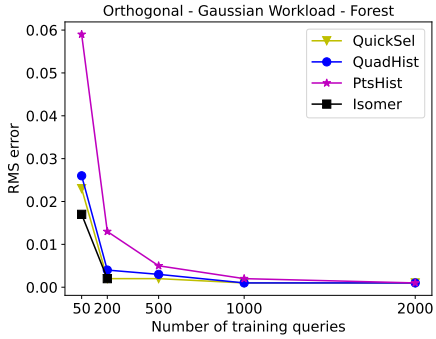

Figure 44: RMS Error vs. training size on Gaussian workload of Forest.

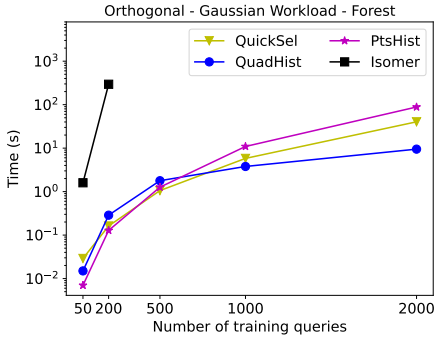

Figure 45: Training time vs. training size on Gaussian workload of Forest.

	Training	Isomer				QuickSel				QuadHist				PtsHist			
	Size	50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX
Data-driven	50	1.053	1.818	3.204	6.061	1.054	1.5	**2.566**	3.8	1.227	5.046	7.659	16.303	1.123	5.575	22.531	24.148
	200	1.009	1.069	**1.152**	1.95	1.007	1.064	1.191	2.056	1.014	1.179	1.324	2.492	1.026	1.215	1.565	2.416
	500	-	-	-	-	1.005	1.043	**1.112**	1.763	1.004	1.053	1.161	1.232	1.008	1.107	1.195	1.454
	1000	-	-	-	-	1.003	1.028	**1.088**	1.783	1.003	1.046	1.156	1.190	1.005	1.060	1.285	2.285
	2000	-	-	-	-	1.003	1.029	1.177	1.477	1.002	1.025	**1.042**	1.066	1.003	1.040	1.190	2.200
Random	50	1.069	3.448	13.991	74.176	1.098	2.452	16.736	487.752	1.055	1.838	**3.112**	6.167	1.178	2.363	3.121	4.712
	200	1.017	1.963	9.026	28.671	1.019	2.466	8.963	21.521	1.025	1.992	3.300	7.702	1.039	2.351	**3.014**	3.688
	500	-	-	-	-	1.012	1.802	2.778	6.53	1.012	1.403	**2.652**	2.969	1.032	1.976	3.422	4.130
	1000	-	-	-	-	1.009	2.226	5.674	33.054	1.008	1.310	**2.059**	2.467	1.020	1.547	7.818	10.350
	2000	-	-	-	-	1.007	1.917	11.936	68.928	1.006	1.372	2.782	3.025	1.013	1.548	**2.056**	2.757
Gaussian	50	1.039	1.255	**1.543**	1.643	1.049	1.538	4.243	14.156	1.049	1.934	4.515	11.638	1.178	2.363	3.121	4.712
	200	1.007	1.156	**1.226**	1.327	1.005	1.112	1.329	1.971	1.012	1.177	1.624	1.649	1.031	1.550	2.650	5.247
	500	-	-	-	-	1.004	1.061	**1.283**	1.632	1.007	1.132	1.357	1.553	1.013	1.183	1.477	1.819
	1000	-	-	-	-	1.004	1.192	1.699	2.259	1.004	1.075	**1.337**	1.342	1.007	1.172	1.376	1.506
	2000	-	-	-	-	1.004	1.087	1.749	1.962	1.003	1.070	**1.113**	1.202	1.003	1.072	1.148	1.237

Table 3: Q-error over Forest. The bold numbers are those ranking the smallest in 99th Q-error.

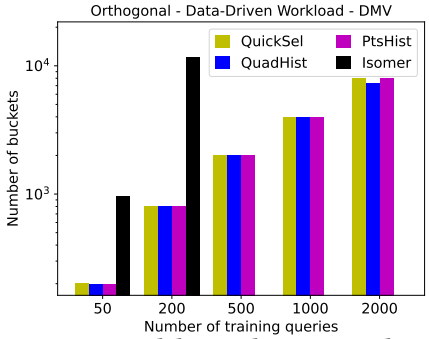

Figure 46: Model complexity over data-driven workload of DMV.

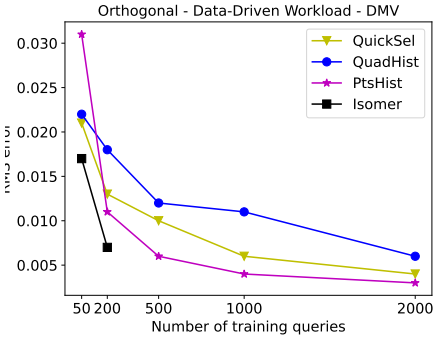

Figure 47: RMS Error vs. training size on data-driven workload of DMV.

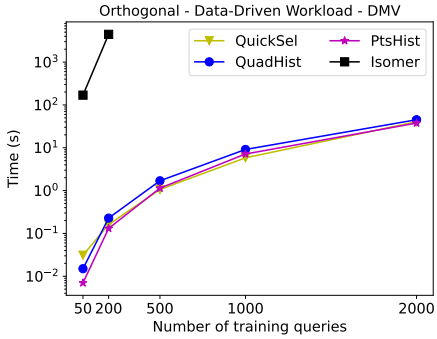

Figure 48: Training time vs. training size on data-driven workload of DMV.

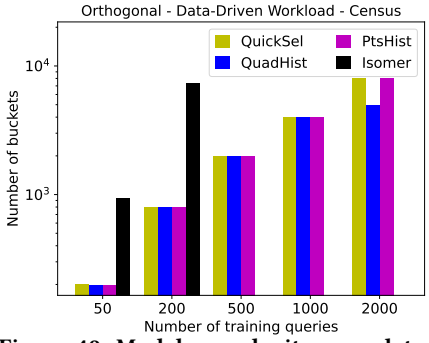

Figure 49: Model complexity over data-driven workload of Census.

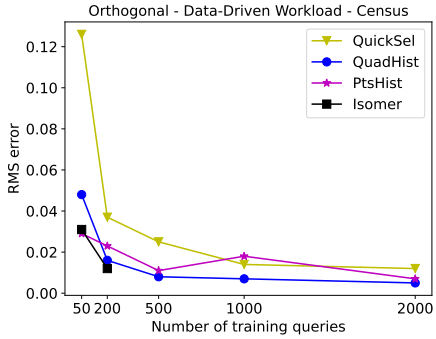

Figure 50: RMS Error vs. training size on data-driven workload of Census.

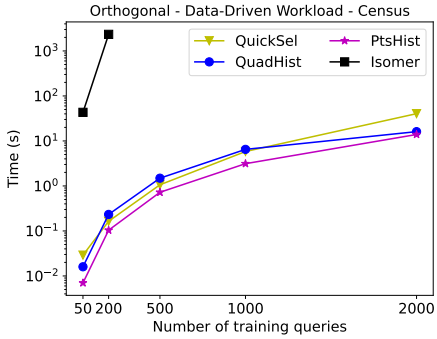

Figure 51: Training time vs. training size on data-driven workload of Census.

	Training Size	Isomer				QuickSel				QuadHist				PtsHist			
		50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX
Data-driven	50	1.074	2.038	6.516	11.946	1.097	2.265	4.301	6.202	1.148	2.783	3.252	5.253	1.098	2.148	**2.91**	3.126
	200	1.026	1.981	4.268	8.21	1.059	2.844	6.101	9.01	1.096	2.577	3.842	4.898	1.043	1.503	**1.959**	2.023
	500	-	-	-	-	1.048	1.988	3.816	4.269	1.061	2.333	3.420	4.748	1.035	1.388	**1.759**	1.965
	1000	-	-	-	-	1.025	1.581	2.101	5.421	1.038	1.648	3.270	8.099	1.022	1.320	**1.379**	1.627
	2000	-	-	-	-	1.021	1.313	2.194	18.025	1.024	1.382	1.962	4.204	1.016	1.289	**1.566**	1.831

Table 4: Q-error over DMV. The bold numbers are those ranking the smallest in 99th Q-error.

	Training Size	Isomer				QuickSel				QuadHist				PtsHist			
		50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX	50th	95th	99th	MAX
Data-driven	50	1.085	2.356	**4.429**	4.502	1.165	5.983	8.558	31.893	1.208	14.575	20.680	28.434	1.084	2.067	4.835	4.843
	200	1.01	1.448	3.094	4.5	1.16	2.221	6.57	10.669	1.055	2.689	6.258	7.847	1.061	1.966	**2.365**	2.404
	500	-	-	-	-	1.094	2.639	8.743	14.788	1.023	1.403	**1.774**	2.107	1.044	1.686	2.137	2.265
	1000	-	-	-	-	1.063	1.835	4.178	6.31	1.014	1.329	1.993	2.102	1.042	1.497	**1.609**	1.664
	2000	-	-	-	-	1.021	1.61	7.276	20.405	1.008	1.280	1.795	1.893	1.018	1.382	**1.512**	1.576

Table 5: Q-error over Census. The bold numbers are those ranking the smallest in 99th Q-error.