Learning to be a Statistician: Learned Estimator for Number of Distinct Values

Renzhi Wu*
Georgia Institute of Technology
renzhiwu@gatech.edu

Bolin Ding
Alibaba Group
bolin.ding@alibaba-inc.com

Xu Chu
Georgia Institute of Technology
xu.chu@cc.gatech.edu

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

Xiening Dai
Alibaba Group
xiening.dai@alibaba-inc.com

Tao Guan, Jingren Zhou
Alibaba Group
{tony.guan,jingren.zhou}@alibaba-inc.com

ABSTRACT

Estimating the number of distinct values (NDV) in a column is useful for many tasks in database systems, such as columnstore compression and data profiling. In this work, we focus on how to derive accurate NDV estimations from random (online/offline) samples. Such efficient estimation is critical for tasks where it is prohibitive to scan the data even once. Existing sample-based estimators typically rely on heuristics or assumptions and do not have robust performance across different datasets as the assumptions on data can easily break. On the other hand, deriving an estimator from a principled formulation such as maximum likelihood estimation is very challenging due to the complex structure of the formulation. We propose to formulate the NDV estimation task in a supervised learning framework, and aim to learn a model as the estimator. To this end, we need to answer several questions: i) how to make the learned model workload agnostic; ii) how to obtain training data; iii) how to perform model training. We derive conditions of the learning framework under which the learned model is *workload agnostic*, in the sense that the model/estimator can be trained with synthetically generated training data, and then deployed into any data warehouse simply as, *e.g.*, user-defined functions (UDFs), to offer efficient (within microseconds on CPU) and accurate NDV estimations for *unseen tables and workloads*. We compare the learned estimator with the state-of-the-art sample-based estimators on nine real-world datasets to demonstrate its superior estimation accuracy. We publish our code for training data generation, model training, and the learned estimator online for reproducibility.

PVLDB Reference Format:
Renzhi Wu, Bolin Ding, Xu Chu, Zhewei Wei, Xiening Dai, Tao Guan, Jingren Zhou. Learning to be a Statistician: Learned Estimator for Number of Distinct Values. PVLDB, 15(2): 272 - 284, 2022.
doi:10.14778/3489496.3489508

*Work done at Alibaba Group.

Proceedings of the VLDB Endowment, Vol. 15, No. 2 ISSN 2150-8097.
doi:10.14778/3489496.3489508

1 INTRODUCTION

Estimating number of distinct values (NDV), also known as cardinality estimation, is a fundamental problem with numerous applications [30, 33, 46, 48]. It has been extensively studied in many research communities including databases [22, 30], networks [25, 48], bioinformatics [46], and statistics [18, 31, 50].

The methods for estimating NDV in the absence of an index can be classified into two categories: *sketch based methods* and *sampling based methods* [45]. Sketch based methods scan the entire dataset once, followed by sorting/hashing rows, and create a sketch that is used to estimate NDV [33]. Sampling based methods estimate NDV using statistics from a small sample, without needing to scan the entire dataset. Generally, sketch based methods give more accurate estimation, but scanning and hashing the entire dataset can be prohibitively expensive in large data warehouses. Hashing techniques such as probabilistic counting help alleviate the memory requirements but still requires a full scan of the table. When a full scan is not possible or the computation cost of a full scan is not affordable, sampling based methods are the only remaining alternatives which exam only a very small fraction of the table, *i.e.*, a sample, and thus scale well with increasing data set.

In this paper, we focus on the problem of accurately estimating the number of distinct values from samples of large tables.

The first challenge is that, unlike some other statistical parameters, such as means and histograms, which can be accurately computed from small random samples, accurate NDV estimation from small samples has been proved to be an extremely difficult task (*e.g.*, with theoretical lower bounds of error given in [22]).

We can formulate sample-based NDV estimation using a principled method for estimating unknown parameters, the *maximum likelihood estimation* (MLE) [19]. An MLE estimator is *workload-agnostic*: it is derived (analytically) before we see the real workloads. It solves an optimization problem, which maximizes the likelihood of observing a specific random sample, and gives an NDV estimation with desirable properties such as *consistency* and *efficiency*. However, the remaining steps are challenging (if not impossible): how to express the likelihood function and the optimization problem in a compact way, and how to (even approximately) solve it.

Due to the above challenges, this paper proposes and studies a more fundamental question: *whether it is possible to train a workload-agnostic machine learning model to approximate principled statistical estimators such as MLE estimators*, with the training set synthetically generated from a training distribution calibrated based on the

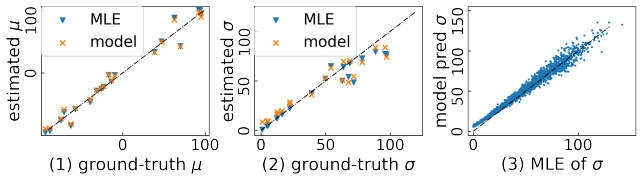

Figure 1: Evaluating trained models to estimate mean μ and STD σ of Gaussian distributions: (1) Estimated μ via MLE and the trained model v.s. ground-truth μ on 20 data points. (2) Estimated σ via MLE and the trained model v.s. ground-truth σ on 20 data points. (3) Model predicted σ v.s. MLE estimation of σ in the closed form on 2000 data points.

properties of our estimation task, such that the learned model can be used on unseen workloads.

Learning to be a Statistician: a Toy Example

It is traditionally the statistician's job to derive a closed-form or numerical estimator as a function that takes an observed sample or sample statistics as the input for estimating unknown parameters of a statistical model. On the other hand, it is known that any continuous function can be approximated by a neural network (with enough nodes and layers) [34], which require three elements: a training set A, a class of model architectures $\mathcal{H}$, and a training algorithm. Thus, a natural question is whether we can train a neural network in a *workload-agnostic way* (without seeing any real dataset during training) to approximate, *e.g.*, an MLE estimator. Following are two examples with positive answers to this question.

Learning MLE of mean and STD. Let's consider the mean μ and standard deviation (STD) σ of a Gaussian distribution. For a size-k sample drawn from this distribution, $\mathbf{S} = \{v_1, \ldots, v_k\}$, the MLE estimations of μ and σ are known to have closed forms: $\mu^{\text{MLE}} = \sum_i v_i/k$ and $\sigma^{\text{MLE}} = \sqrt{\sum_i (v_i - \mu^{\text{MLE}})^2/k}$, respectively. Assume that we do not know the concrete form of μ^{MLE} and σ^{MLE}. How should we train machine learning models to approximate them?

We first need to prepare the training data. From a specific Gaussian distribution $\mathcal{N}(\mu, \sigma)$, we draw a size-k sample $\mathbf{S}$, and add *one* training data point $(\mathbf{S}, \mu)$ to the training set A_1 for mean estimation, and *one* data point $(\mathbf{S}, \sigma)$ to A_2 for STD estimation. Note that, here, each $\mathbf{S}$ is a k-dim feature vector, and each μ and σ are the (ground-truth) labels to be predicted for mean and STD, respectively. In the outer loop, we need to try different combinations of μ and σ to add more training data points into A_1 and A_2. In this example, we generate 10^5 pairs of (μ, σ), where each μ is randomly picked from a Gaussian distribution $\mathcal{N}(0, 100)$ and each σ is picked from a folded Gaussian distribution (*i.e.*, $\sigma = |z|$ where $z \sim \mathcal{N}(0, 100)$). Thus, we will have 10^5 training data points in each of A_1 and A_2.

We train a regression model (multi-layer perception with one hidden layer of size k and mean squared loss) using the training set A_1 for estimating μ and another model (with the same architecture) on A_2 for estimating σ, both with $\mathbf{S}$ as feature.

We evaluate the trained estimator on a completely different test set. We generate pairs (μ, σ) with each $\mu \sim U(-100, 100)$ and $\sigma \sim U(0, 100)$ from uniform distributions. For each (μ, σ), we generate a test data point $(\mathbf{S}, \mu)$ and a point $(\mathbf{S}, \sigma)$ for a size-k sample $\mathbf{S}$ from $\mathcal{N}(\mu, \sigma)$. The results are reported in Figure 1 (where we use $k = 100$). Figures 1(1)-(2) show that the estimations of mean/STD given by the trained model have very similar performance as those by the closed-form MLE estimations. The trained model and MLE even make the same mistakes for some test points with non-trivial errors. Figure 1(3) compares σ^{MLE} and the learned estimator on more data points for STD estimation, for which the MLE estimator has a more complex form than a linear model for the mean estimation.

Our Contributions and Solution Overview

In this paper, we will formalize and investigate the learning framework illustrated in the above example, but for a more challenging task, *sample-based NDV estimation*. While the empirical evidence for mean/STD estimation sheds light on the possibility of approximating MLE estimators with trained models, there are several key questions to be resolved for general estimation tasks: i) how to generate training data and what features should be extracted for learning the model; ii) what model architecture, loss function, and regularization should be used; iii) with the choice in i) and ii), whether the trained estimator approximates MLE and performs in a robust and workload-agnostic way.

• *MLE formulation.* We first formulate NDV estimation as an MLE problem. Assuming that a data column is generated from some prior distribution and a sample $\mathbf{S}$ is drawn uniformly at random from the column, we observe the sample profile, *i.e.*, a vector $f = (f_j)_{j=1,\ldots}$ where f_j is the number of distinct values with frequency j in $\mathbf{S}$. For example, for $\mathbf{S} = \{\text{a}, \text{a}, \text{a}, \text{b}, \text{b}, \text{b}, \text{c}, \text{c}\}$, we have $f_1 = 0$, $f_2 = 1$ ('c' has frequency 2), and $f_3 = 2$ ('a' and 'b'). For an MLE estimation, we aim to find the value of NDV, D, such that the probability of observing f conditioned on NDV $= D$ is maximized. Although it is difficult to solve (even approximately) the MLE formulation, it will guide the design and analysis of our machine learning model.

• *Learning estimator that approximates MLE.* With the sample profile f as a feature and the ratio error between the estimate and the true NDV as the training loss, we have a skeleton of the learning framework. Unlike learning MLE for mean and STD where the training data preparation is trivial because each dimension of the feature space (v_i in $\mathbf{S}$) can be independently generated, in NDV estimation, however, different dimensions of the sample profile (f_i in f) are correlated, conditioned on NDV; thus, it is more difficult to characterize the distribution needed for preparing the training set. After carefully investigating the probability distribution the model learns based on our choice of feature and loss and comparing the learned probability distribution with the one in our MLE formulation, we derive several precise properties to characterize the training distribution we need in order to make the learned estimator approximate an MLE estimator.

• *Efficient training data generation.* We design an efficient algorithm to generate training data points. To generate one data point, we need to first generate a data column (in a compact way). And instead of drawing a sample explicitly, we directly generate the sample profile from its distribution determined by the data column. In our experiment, 10^5-10^6 data points suffice to train the estimator.

• *Instance-wise negative results and model regularization.* It is important to note that the negative result about sample-based NDV estimation in [22] also hold for "learned estimators". However, the

negative result in [22] is a global one, in the sense that there exist *hard instances* of data columns whose NDVs are difficult to be estimated. However, the hope is that, upon the observation of the sample profile, it is possible to infer that the underlying data column is not a hard instance and an estimate with error lower than the global lower bound can be expected. We first derive a new instance-wise negative result, *i.e.*, a lower bound of the estimation error upon the observation of a sample profile. We use this negative result to regularize the training of our model. More specifically, we propose a regularization method that encourages the learned estimator to achieve just the instance-wise lower bound of estimation error, instead of always minimizing the estimation error to be zero (which is an impossible goal due to global negative result). Its effectiveness will be verified via experimental studies.

• *Deployment and experiments.* The learned estimator, once trained, can be deployed in any data warehouse without the need of retraining or tuning. We publish our estimator online [43] as a trained neural network that takes a sample/sample profile of a column as the input feature and outputs an estimated NDV of that column. It is compared with the state-of-the-art sample-based NDV estimators and is shown to outperform them on nine different real datasets. Note, again, that none of these datasets is used to train our estimator, and all the training data is generated synthetically.

Related Work

Learned cardinality estimation. There have been a long line of existing works on learning a model to estimate cardinality or selectivity of a query [15, 26, 27, 41, 42, 44, 60]. In general, they can be divided into two types [57]. The first type uses query as the primary feature and training data comes from the records of query executions. For example, [27] uses query as features and applies tree-based ensembles to learn the selectivity of multi-dimensional range predicates. This type of method typically requires executing a huge number of queries to obtain enough training data [57]. There are efforts to alleviate this issue: [26] proposes to reduce model construction cost by incrementally generating training data and using approximate selectivity label. Generally, this type of methods work well when future queries follow the same distribution and similar templates (*e.g.*, conjunction of range and categorical predicates) as the training data [60]. The second type (for example [35] and [60]) builds a model to approximate the joint distribution of all attributes in a table. This type of methods require re-training in case of data or schema update. In dynamic environments with unseen datasets and workloads coming, both type of methods require huge efforts of accessing the new dataset and retraining [57].

Although we also learn a model to estimate NDV/cardinality, our method is completely different from the above methods. Our method is a sample-based approach and we use features from sample rows (drawn from tables or query results) instead of queries. More importantly, our method is workload agnostic (only needing to be trained once) and can be applied to any dynamic workload.

Sample-based NDV estimation. Existing sampling based methods are typically constructed based on heuristics or derived by making certain assumptions. For example, there are estimators derived by assuming infinite population size [21], certain condition of skewness [52], and certain distribution of the data [47]. GEE estimator [22] is constructed to match a (worst-case) lower bound of estimation error in [22]. We discuss these existing approaches in more details in Section 2.2. These approaches are not robust especially on datasets where their assumptions about data distribution break, as we will show in experiments in Section 6.

Sketch-based NDV estimation. Sketch-based methods (*e.g.*, Hyperloglog [28]) scan the entire dataset once and keep a memory-efficient sketch that is used to estimate NDV. These methods are able to produce highly accurate NDV estimation. For example, the expected relative error of HyperLogLog [28] is about $1.04/\sqrt{m_{\text{bytes}}}$, where m_{bytes} is the number of bytes used in HyperLogLog; With 10K bytes, the relative error of NDV estimation can be as small as 1%-2%. See survey [33] for a comprehensive review.

Sample v.s. Sketch. When one scan of the data is affordable, sketch-based estimators are preferable to sample-based ones. In scenarios where a full scan of the data is (relatively) too expensive, sample-based methods are preferable. For example, when NDV estimation in a column is used by the query optimizer to generate a good execution plan for a SQL query, a high-quality sample-based estimation is preferable, as the query is almost processed after a full scan. Another example is approximate query processing [24], where offline samples are used to provide approximate answers to analytical (*e.g.*, NDV) queries over voluminous data with interactive speed. In short, sketch-based estimators and sample-based methods are two orthogonal lines of research with different applications.

Paper Organization

In Section 2, we provide the preliminaries including an MLE-based formulation for NDV estimation, and review some representative estimators. In Section 3, we provide an overview of our learning-to-estimate framework; we analyze and derive properties of the training distribution needed to make the learned model approximate MLE. In Section 4, we give details about training data generation and model architecture; we also introduce our new instance-wise negative result about sample-based NDV estimation, and how to regularize the model accordingly. Section 5 gives a brief introduction on how to use our learned estimator that is available online. We evaluate our estimator and compare it with the state of the arts experimentally and report the results in Section 6.

2 PRELIMINARIES

We focus on a specific *data column* $\mathbf{C}$ of a table with N rows. N is called the *population size*. Let D be the *number of distinct values* (NDV) in $\mathbf{C}$. When calculating NDV of the column $\mathbf{C}$, we consider $\mathbf{C}$ as a (multi)set of values from some (possibly infinite) domain Ω.

Problem statement: estimating NDV from samples. We want to estimate D from a random sample $\mathbf{S} \subseteq \mathbf{C}$ of n tuples drawn uniformly at random from $\mathbf{C}$. Let $r = n/N$ be the sampling rate. We assume N, or equivalently r, is observed.

We first formally define two important notations.

Frequency. The *frequency* of a value $i \in \Omega$ in a column $\mathbf{C}$, or a sample $\mathbf{S}$, is the number of times it appears in $\mathbf{C}$, or $\mathbf{S}$, denoted as N_i, or n_i, respectively. By definitions, we have $\sum_i N_i = N$ and $\sum_i n_i = n$, and the NDV in $\mathbf{C}$ is $D = |\{i \in \Omega \mid N_i > 0\}|$.

Profile. In order to calculate NDV, it is sufficient to consider the *profile* of $\mathbf{C}$, denoted as $F = (F_j)_{j=1,\ldots,N}$, where $F_j = |\{i \in$

$\Omega \mid N_i = j\}|$ is the number of distinct values with frequency j in $\mathbf{C}$; similarly, the profile of a random sample $\mathbf{S}$ is $f = (f_j)_{j=1,\ldots,n}$ where $f_j = |\{i \in \Omega \mid n_i = j\}|$ is the number of distinct values with frequency j in $\mathbf{S}$. By definitions, the NDV in $\mathbf{C}$ is $D = \sum_{j>0} F_j$ and the population size is $N = \sum_j jF_j$; The NDV in sample $\mathbf{S}$ is $d = \sum_{j>0} f_j$ and the sample size is $n = \sum_j jf_j$.

2.1 An MLE-based Formulation.

Estimating NDV from random samples can be formulated as a *maximum likelihood estimation* (MLE) problem, which is commonly used to estimate unknown parameters of a statistical model, with desirable properties such as consistency and efficiency [19]. The estimated value (or the solution to an MLE problem) maximizes the probability of the observed data generated from this model.

We assume that the column $\mathbf{C}$ with profile F is drawn from some prior probability distribution. A uniformly random sample $\mathbf{S}$ with profile f is then drawn from $\mathbf{C}$. An MLE-based formulation for NDV estimation can be derived based on the *observed* profile in the sample $\mathbf{S}$, *i.e.*, the sample profile f, and the observed population size N (or equivalently, sampling rate $r = n/N$). We estimate D as the one that maximizes the probability of observing f and N.

$$D^{\mathsf{MLE}} = \arg\max_D \mathbb{P}(f, N \mid D) = \arg\max_D \sum_F \mathbb{P}(f, N \mid F)\,\mathbb{P}(F \mid D)\,.$$

Define $\mathcal{F}(D,N) = \{F \mid \sum_{j>0} F_j = D \text{ and } \sum_{j>0} j \cdot F_j = N\}$ to be all the *feasible* profile configurations with NDV equal to D and population size equal to N. For $F \in \mathcal{F}(D,N)$, we have $\mathbb{P}(f, N \mid F) = \mathbb{P}(f \mid F)$; and for $F \notin \mathcal{F}(D,N)$, we have $\mathbb{P}(f, N \mid F) = 0$ and $\mathbb{P}(F \mid D) = 0$. The above formulation can be rewritten as:

$$D^{\mathsf{MLE}} = \arg\max_D \sum_{F \in \mathcal{F}(D,N)} \mathbb{P}(f \mid F)\,\mathbb{P}(F \mid D)\,.$$

The estimator D^{MLE} is to be used on unknown columns; so in order to solve the above optimization problem, it is reasonable to assume that the prior distribution of F is uniform, in the sense that every possible profile in $\mathcal{F}(D,N)$ appears with equal probability, *i.e.*, $\mathbb{P}(F \mid D) = 1/|\mathcal{F}(D,N)|$ for every $F \in \mathcal{F}(D,N)$. Under this assumption, we want to solve the following one for D^{MLE}:

$$D^{\mathsf{MLE}} = \arg\max_D \frac{1}{|\mathcal{F}(D,N)|} \sum_{F \in \mathcal{F}(D,N)} \mathbb{P}(f \mid F)\,. \tag{1}$$

We can also interpret the MLE-based formulation (1) as follows. After observing the sample profile f and the population size N, we estimate D as D^{MLE} which maximizes the average probability of generating f from a feasible profile $F \in \mathcal{F}(D,N)$. Solving (1), however, is difficult even approximately, and thus this formulation has not been applied for estimating NDV yet.

A natural question is whether it benefits to use sample $\mathbf{S}$, instead of sample profile f, as observed data to derive MLE and as features in our learning framework. We defer a formal analysis on why using sample is not more advantageous to a technical report [58] due to space limit. In fact, most existing estimators (as we show next) also use sample profile instead of sample to estimate NDV.

2.2 Existing Estimators

There have been a long line of works on estimating NDV from random samples. We review some representative ones as follows.

- A problem related to the one in (1) is *profile maximum likelihood estimation* (PML) [23, 32, 50], which chooses F that maximizes the probability of observing f of the randomly drawn $\mathbf{S}$. Define:

$$F^{\mathsf{PML}} = \arg\max_F \mathbb{P}(f \mid F) \quad \text{and} \quad D^{\mathsf{PML}} = \textstyle\sum_{j>0} F_j^{\mathsf{PML}}. \tag{2}$$

There have been works on finding approximations to F^{PML} [23, 50], which can be in turn used to obtain an approximate version of D^{PML} in (2), although, in general, $D^{\mathsf{MLE}} \neq D^{\mathsf{PML}}$.

- Shlosser [52] is derived based on an assumption about skewness: $\mathbf{E}[f_i]/\mathbf{E}[f_1] \approx F_i/F_1$, and performs well when each distinct value appear approximately one time on average [30]. It estimates D as

$$D^{\mathsf{Shlosser}} = d + (f_1 \textstyle\sum_{i=1}^n (1-r)^i f_i) \Big/ (\sum_{i=1}^n ir(1-r)^{i-1} f_i). \tag{3}$$

- Chao [20] approximates the expected NDV, $\mathbf{E}[D]$, in large population for some underlying distribution, and estimates NDV as a lower bound of $\mathbf{E}[D]$ with the population size approaching infinity:

$$D^{\mathsf{Chao}} = d + f_1^2/(2f_2). \tag{4}$$

- GEE [22] is constructed by using geometric mean to balance the two extreme cases for values appearing exactly once in the sample: those with frequency one in $\mathbf{C}$ and drawn into $\mathbf{S}$ with probability r v.s. those with high frequency in $\mathbf{C}$ and at least one copy drawn into $\mathbf{S}$. It is proved to match a theoretical lower bound of ratio error for NDV estimation within a constant factor.

$$D^{\mathsf{GEE}} = \sqrt{1/r} \cdot f_1 + \textstyle\sum_{i=2}^n f_i \tag{5}$$

HYBGEE [22] is a hybrid estimator using GEE for high-skew data and using the smoothed jackknife estimator for low-skew data. AE [22] is a more principled version of HYBGEE with smooth transition from low-skew data to high-skew data. It requires to solve a non-linear equation using, *e.g.*, Brent's method [10].

2.3 Negative Results

The aforementioned lower bound of ratio error for sample-based NDV estimators is given by Charikar *et al.* in [22]. More formally, define the *ratio error* of an estimation $\hat{D}$ w.r.t. the true NDV D to be

$$\text{error}(\hat{D}, D) = \max\{\hat{D}/D, D/\hat{D}\}. \tag{6}$$

It considers in [22] a even larger class of estimators which randomly and adaptively examine n tuples from $\mathbf{C}$. Note that drawing a random sample $\mathbf{S} \subseteq \mathbf{C}$ of n tuples is a special case here. It says that for any such estimator, there exists a choice of column $\mathbf{C}$ such that, with probability at least $\gamma > e^{-n}$, the ratio error is at least

$$\text{error}(\hat{D}, D) \geq \sqrt{\tfrac{N-n}{2n} \ln \tfrac{1}{\gamma}}. \tag{7}$$

3 OVERVIEW OF LEARNING FRAMEWORK

Since analytically solving (1) even approximately is difficult, we propose to formulate the task of deriving D^{MLE} as a supervised learning problem. We first introduce some key elements in the learning model, including the loss function and the design of training dataset, which learns an estimator to approximate (1).

Learning to estimate. In typical traditional estimators such as (3)-(5), the NDV is estimated as a *function* $\hat{D} = \mathsf{h}(f, r)$ of the sample profile f and the sampling rate r, or equivalently a *function* $\hat{D} = \mathsf{h}(f, N)$ of f and the population size N as $r = \sum_{i>0} if_i/N$. As it is difficult to directly derive the function h from a principled formulation such as MLE in (1), we attempt to *learn* the MLE of NDV from a set of training data points $A = \{((f, N), D)\}$, where in each point, (f, N) is the *input feature* and the NDV D is the *label* to be predicted. We use the ratio error defined in (6) to measure the accuracy of estimations, and accordingly, the loss function of the model h is:

$$L(\mathsf{h}(f, N) = \hat{D}, D) = |\log \hat{D} - \log D|^2 = \left(\log \text{error}(\hat{D}, D)\right)^2. \quad (8)$$

Given a hypothesis set $\mathcal{H}$ of estimation functions, the goal is to find $\mathsf{h} \in \mathcal{H}$ with small *empirical loss*:

$$\hat{R}_A(\mathsf{h}) = 1/|A| \sum_{((f,N),D)\in A} L(\mathsf{h}(f, N), D). \quad (9)$$

In order to generate a data point $((f, N), D)$ in the training set A, we first generate a data column $\mathbf{C}$ with profile F according to a *training distribution* $\mathcal{A}$. The NDV D as well as population size N is directly calculated from F. A random sample $\mathbf{S}$ is drawn uniformly at random from $\mathbf{C}$ with sampling rate r. The sample profile f is then obtained from $\mathbf{S}$. The above process is repeated independently multiple times to generate a training set $A = \{((f, N), D)\}$.

Approximating MLE. Intuitively, for a different training distribution $\mathcal{A}$, a different learned estimator h will be derived from the hypothesis set (model architecture) $\mathcal{H}$. Assuming that $\mathcal{H}$ is expressive enough [51], we give some guidelines here on how to choose $\mathcal{A}$ so that the learned h approximates the MLE estimator D^{MLE}.

Let $\mathbb{P}_{\mathcal{A}}(N, D, F, f)$ denote the joint distribution of (N, D, F, f) in the training set A that is generated from the training distribution $\mathcal{A}$ as above. Using the loss function in (8)-(9), *i.e.*, the L2 loss on $\log D$, the trained model h learns the distribution $\mathbb{P}_{\mathcal{A}}(\log D \mid f, N)$ [29]. By minimizing (9), the model tends to emit an output

$$h(f, N) \approx \arg\max_D \mathbb{P}_{\mathcal{A}}(\log D \mid f, N), \quad (10)$$

for a given input (f, N). The approximation sign $\approx$ is because the trained model might not be able to learn the underlying distribution $\mathcal{A}$ of the training set exactly and the accuracy of the approximation is also determined by the hypothesis set $\mathcal{H}$ and the training algorithm. For the term $\mathbb{P}_{\mathcal{A}}(\log D \mid f, N)$ on the right hand side,

$$\mathbb{P}_{\mathcal{A}}(\log D \mid f, N) = 1/\mathbb{P}_{\mathcal{A}}(f \mid N) \cdot \mathbb{P}_{\mathcal{A}}(\log D \mid N)\, \mathbb{P}_{\mathcal{A}}(f \mid \log D, N)$$
$$= \frac{\mathbb{P}_{\mathcal{A}}(\log D \mid \log N)}{\mathbb{P}_{\mathcal{A}}(f \mid N)} \sum_{F\in\mathcal{F}(D,N)} \mathbb{P}(f \mid F)\, \mathbb{P}_{\mathcal{A}}(F \mid D, N) \quad (11)$$

where both equations are from properties of conditional probability.

Consider the following two conditions about the distribution $\mathcal{A}$:

$$\text{i) for any } N,\ \mathbb{P}_{\mathcal{A}}(\log D \mid \log N) = \text{constant}; \quad (12)$$
$$\text{ii) for any } N \text{ and } D,\ \mathbb{P}_{\mathcal{A}}(F \mid D, N) = 1/|\mathcal{F}(D, N)|. \quad (13)$$

Namely, i) for any fixed N, the NDV D distributes uniformly at log scale (or equivalently $\log D$ distributes uniformly) in $\mathcal{A}$; and ii) for any fixed N and D, every feasible profile appears with equal probability. If both are satisfied, the maximizer of (11) (as a function of D) can be written as

$$\arg\max_D \mathbb{P}_{\mathcal{A}}(\log D \mid f, N) = \arg\max_D \frac{\text{constant}}{\mathbb{P}_{\mathcal{A}}(f \mid N)} \sum_{F\in\mathcal{F}(D,N)} \frac{\mathbb{P}(f \mid F)}{|\mathcal{F}(D, N)|}$$
$$= \arg\max_D \frac{1}{|\mathcal{F}(D, N)|} \sum_{F\in\mathcal{F}(D,N)} \mathbb{P}(f \mid F) = D^{\mathsf{MLE}} \quad (14)$$

by putting (12)-(13) back into (11). The second equality in (14) is because $\mathbb{P}_{\mathcal{A}}(f \mid N)$ is a probability term that is independent on D; and the last equality is from the definition of D^{MLE} in (1). Therefore, the learned estimator $\mathsf{h}(f, N)$ approximates D^{MLE} if the training distribution satisfies the two conditions in (12)-(13).

From the above analysis, the learned estimator $\mathsf{h}(f, N)$ approximates (14) which depends on only $\mathcal{F}(D, N)$ and $\mathbb{P}(f \mid F)$: the former is a deterministic finite set, and the latter is a probability distribution depending only on the sampling procedure but not on $\mathcal{A}$; therefore, intuitively, we can expect that $\mathsf{h}(f, N)$ is workload-agnostic, *i.e.*, it generalizes well on unseen data columns, which will be verified later in our experiments reported in Section 6.

About uniformity of F. When introducing the MLE estimator D^{MLE} and analyzing its equivalence to the learned model $\mathsf{h}(f, N)$, the underlying assumption is that the prior distribution of F is uniform. In fact, any targeted prior distribution of F can be plugged into D^{MLE} and accordingly, used as the training distribution $\mathcal{A}$ for h. However, we assume the uniformity purposely to enable the learned estimator generalize to unseen datasets (workload-agnostic). We tried to train an estimator (*Lower Bound (LB)* in Section 6) with the prior of F and $\mathcal{A}$ the same as those in the test datasets. In experiments, we observed that the LB estimator works well only on this particular workload but does not generalize to unseen workloads. In comparison to the LB estimator with the "optimal" prior, our workload-agnostic estimator with the uniform prior has comparable performance on every test table (refer to, *e.g.*, Table 2).

4 LEARNING ESTIMATOR FROM DATA

4.1 Efficient Training Data Generation

As long as the training data satisfies the conditions in (12)-(13), whether the data is synthetic or from real world makes no difference. This makes it possible for us to train the model using synthetically generated data without needing any real-world data.

To generate a training data point $((f, N), D)$, one may first generate a random column $\mathbf{C}$, then draw a random sample $\mathbf{S}$ from $\mathbf{C}$ and calculate sample profile f from $\mathbf{S}$. However, since the population profile F of $\mathbf{C}$ contains all required information to generate a random sample profile f (as we will show in Section 4.1.2), we can directly randomly generate a population profile F and then draw a random sample profile f from $\mathbb{P}_{\mathcal{A}}(f|F)$.

4.1.1 Population profile generation. Generating population profile is to draw samples from the distribution $\mathbb{P}_{\mathcal{A}}(F)$, written as:

$$\mathbb{P}_{\mathcal{A}}(F) = \sum_N \sum_D \mathbb{P}_{\mathcal{A}}(F|N, D)\, \mathbb{P}_{\mathcal{A}}(D|N)\, \mathbb{P}_{\mathcal{A}}(N). \quad (15)$$

We can draw samples from $\mathbb{P}_{\mathcal{A}}(F)$ by repeating the following: sample a N from $\mathbb{P}_{\mathcal{A}}(N)$, sample a D from $\mathbb{P}_{\mathcal{A}}(D|N)$, and sample a F from $\mathbb{P}_{\mathcal{A}}(F|N, D)$, with $\mathbb{P}_{\mathcal{A}}(D|N)$ and $\mathbb{P}_{\mathcal{A}}(F|N, D)$ satisfying the conditions in (12)-(13). In order to ensure the training data to be diverse, we want to have N cover a big range of magnitude, so we set the

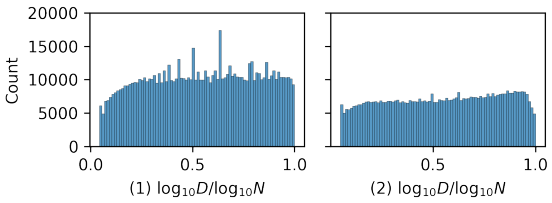

Figure 2: Frequency distribution of $\log D/\log N$.

distribution of N to be uniform at log scale, *i.e.*, $\log_{10} N \sim U(0, B)$ where B is a constant specifying the maximum population size. We select $B = 9$ according to the memory limit of our machine.

Generating F from $\mathbb{P}_{\mathcal{A}}(F|N, D)$ is, however, non-trivial. Given population size N and population NDV D, drawing a population profile from $\mathbb{P}_{\mathcal{A}}(F|N, D)$ is equivalent to randomly sampling an element under i) the uniform-distribution constraint (*i.e.*, every feasible F has equal probability to be drawn) from set $\mathcal{F}(D, N)$ which is specified by the two constraints ii) $\sum_{j>0} F_j = D$ and iii) $\sum_{j>0} jF_j = N$. Designing such a sampling procedure that satisfies the three constraints simultaneously is very challenging.

If the dimensionality of F can be given, the sampling procedure is easier. Consider an alternative form of $\mathbb{P}_{\mathcal{A}}(F)$:

$$\mathbb{P}_{\mathcal{A}}(F) = \sum_N \sum_M \mathbb{P}_{\mathcal{A}}(F|N, M)\, \mathbb{P}_{\mathcal{A}}(M|N)\, \mathbb{P}_{\mathcal{A}}(N). \tag{16}$$

This indicates an alternative procedure: sample a N from $\mathbb{P}_{\mathcal{A}}(N)$, sample a M from $\mathbb{P}_{\mathcal{A}}(M|N)$, and sample a F from $\mathbb{P}_{\mathcal{A}}(F|N, M)$. In this procedure, dimensionality M is given when sampling F from distribution $\mathbb{P}_{\mathcal{A}}(F|N, M)$. The distributions $\mathbb{P}_{\mathcal{A}}(D|N)$ and $\mathbb{P}_{\mathcal{A}}(F|N, D)$ are now implicitly induced from $\mathbb{P}_{\mathcal{A}}(F)$. To see why it is the case, when $\mathbb{P}_{\mathcal{A}}(F)$ is derived as in (16), the joint distribution $\mathbb{P}_{\mathcal{A}}(F, N, D)$ is determined, because, by definitions, N and D are deterministically dependent on F, *i.e.*, $\mathbb{P}_{\mathcal{A}}(F, N, D) = \mathbb{P}_{\mathcal{A}}(F)$ if $N = \sum_i iF_i$ and $D = \sum_i F_i$, and $\mathbb{P}_{\mathcal{A}}(F, N, D) = 0$ otherwise. $\mathbb{P}_{\mathcal{A}}(N, D)$ and $\mathbb{P}_{\mathcal{A}}(N)$ are just marginal distributions of $\mathbb{P}_{\mathcal{A}}(F, N, D)$. Therefore, by the definition of conditional probabilities, we can obtain $\mathbb{P}_{\mathcal{A}}(D|N)$ as $\mathbb{P}_{\mathcal{A}}(N, D)\,/\mathbb{P}_{\mathcal{A}}(N)$ and $\mathbb{P}_{\mathcal{A}}(F|N, D)$ as $\mathbb{P}_{\mathcal{A}}(F, N, D)\,/\mathbb{P}_{\mathcal{A}}(N, D)$.

We plan to design the two distributions $\mathbb{P}_{\mathcal{A}}(M|N)$ and $\mathbb{P}_{\mathcal{A}}(F|N, M)$ to make the induced $\mathbb{P}_{\mathcal{A}}(D|N)$ and $\mathbb{P}_{\mathcal{A}}(F|N, D)$ approximately satisfy the conditions in (12)-(13). In the following, we first intuitively specify the form of $\mathbb{P}_{\mathcal{A}}(M|N)$ and $\mathbb{P}_{\mathcal{A}}(F|N, M)$, then derive an efficient sampling algorithm to generate F, and show that the conditions in (12)-(13) are approximately satisfied in the end.

Achieving (approxiamte) uniformity in sample. Intuitively, M is the highest number of times that a value can appear in population. We draw M uniformly at log scale *i.e.* $\log_{10} M \sim U(0, B)$. Note that M is the dimensionality of the space where we draw F and it is different from the actual length of F (maximum l with $F_l > 0$), so F_M can be zero. Thus, M and N are independent (with $\log_{10} N \sim U(0, B)$), and $\mathbb{P}_{\mathcal{A}}(M|N)$ is identical to $\mathbb{P}_{\mathcal{A}}(M)$.

Given N and M, similarly we can have a feasible configuration set for F: $\mathcal{F}'(N, M) = \{F|\sum_{i=1}^{M} iF_i = N\}$. To make $\mathbb{P}_{\mathcal{A}}(F|D, N)$ approximately uniform (required in (13)), intuitively, we need to make the distribution of F as diffusive as possible, so we also draw F uniformly from $\mathcal{F}'(N, M)$. Notice that compared with $\mathcal{F}(N, D)$, $\mathcal{F}'(N, M)$ only has one constraint and the dimensionality of F is given. This make it much easier to design a sampling procedure.

Let SF_i denote $\sum_i^M F_i$, then $F_i = SF_i - SF_{i+1}$. Apparently, SF has a one-to-one mapping relationship with F, so drawing each feasible F with equal probability is equivalent to drawing each feasible SF with equal probability. The feasible set for SF is $Q(N, M) = \{SF \mid \sum_{i=1}^{M} SF_i = N;\ SF_i \geq SF_{i+1}\ \forall\ i\}$. Without the constraint $SF_i \geq SF_{i+1}$, the problem of generating SF so that each feasible SF has equal probability to be generated is known as the *random fixed sum* problem, and there are existing efficient algorithms to solve it in $O(M \log M)$ [9, 14]. The constraint $SF_i \geq SF_{i+1}$ can be easily satisfied afterwards by reassigning SF to be its sorted version. Once a SF is generated, the corresponding F can be obtained.

To summarize, the process of generating one population profile F is as follows: draw N from $\log_{10} N \sim U(0, B)$, draw M from $\log_{10} M \sim U(0, B)$, draw SF from $Q(N, M)$ using algorithms for the random fixed sum problem, and obtain F by $F_i = SF_i - SF_{i+1}$.

As discussed above, $\mathbb{P}_{\mathcal{A}}(F|N, D)$ and $\mathbb{P}_{\mathcal{A}}(\log D|\log N)$ can be induced from the distribution $\mathbb{P}_{\mathcal{A}}(F)$, and thus also from $\mathbb{P}_{\mathcal{A}}(F|N, M)$ and $\mathbb{P}_{\mathcal{A}}(M|N)$. The uniformity of $\mathbb{P}_{\mathcal{A}}(F|N, D)$ and $\mathbb{P}_{\mathcal{A}}(\log D|\log N)$ is still not strictly guaranteed. For $\mathbb{P}_{\mathcal{A}}(F|N, D)$ and $\mathbb{P}_{\mathcal{A}}(F|N, M)$, however, the sizes of the supporting sets $\mathcal{F}$ and $\mathcal{F}'$, respectively, are both extremely big and it can be shown the fraction of F that we can sample with our best effort ($\sim 10^9$ elements of F) is at the scale of 10^{-3991}. This leads to the fact that the samples we drawn are extremely sparse in the space of all possible F and each F can at most appear once in the samples. Therefore, the empirical distribution of $\mathbb{P}_{\mathcal{A}}(F|N, D)$ in training data will be close to uniform. For $\mathbb{P}_{\mathcal{A}}(\log D|\log N)$, we empirically show its uniformity in Figure 2. For N and D drawn from $\log_{10} N \sim U(0, B)$ and $\log_{10} D \sim U(0, \log_{10} N)$, the ratio $\log D/\log N$ should follow the uniform distribution $U(0, 1)$ and we show the frequency distribution of $\log D/\log N$ in Figure 2(1); we obtain the N and D of our generated profiles and plot the frequency distribution of $\log D/\log N$ – as shown in Figure 2(2), it is also roughly uniformly distributed.

4.1.2 Sample profile generation. To draw a sample, we need to know the sampling rate r. The conditions in (12)-(13) have no constraint on the distribution of r. To ensure the training data to be diverse, we would like to have r cover a big range of magnitude. Therefore, we draw sampling rate r uniformly at log scale by $\log_{10} r \sim U(-B', -1)$ where we select $B' = 4$ as 10^{-4} is a small enough sampling rate in practice.

Instantiating the population/column from F and then performing random sampling to get a sample S and then sample profile f is of complexity $O(N)$. We propose to directly perform sampling from F with a complexity of $O(D)$. When performing sampling, for a value appeared K times in population, the number of times k it appears in sample follows a binomial distribution $\mathbb{P}(k) = \binom{K}{k} r^k (1-r)^{K-k}$ where r is the sampling rate. By performing a binomial toss for every distinct value in population, we obtain the number of times each value appear in sample and sample profile f can be calculated. Accordingly, the complexity of generating a sample profile is $O(D)$.

4.1.3 Diversity training data. To further improve the generalization ability of the trained model, we diversify the profiles we draw from $\mathcal{F}'$ by incorporating some human knowledge. This is in the same

spirit as that one may incorporate human knowledge to diversify an image dataset by operations such as rotation to improve the generalization ability of models trained on the dataset [53].

We diversify our training data based on the following intuition: each F can be seen as a point in high dimensional space with F_i being its coordinate at the ith dimension; we want to enlarge the supporting region of the points (*i.e.*, the region that the points span) so that the trained model generalizes better, since machine learning models are better at interpolation than extrapolation [59].

The most ideal way to increase the support region of the profiles is to have some data points with a much bigger population size. In this way, F_i at every dimension can be very big yielding a bigger support region. However, due to hardware limit in practice we are not able to achieve this, so we choose to randomly increase one F_i along one single random dimension for each F. Specifically, for each profile F generated by the method in Section 4.1.1, a new component F' is added to F to obtain the final profile $F'' = F + F'$ where F' contains only one positive value $F'_{i_\mathrm{p}} = D'$ and $F'_{i \neq i_\mathrm{p}} = 0; D'$ and i_p are randomly generated by $\log_{10} N' \sim U(0, B)$, $\log_{10} D' \sim U(0, \log_{10} N')$, and $i_\mathrm{p} = \lfloor N'/D' \rceil$. Since F' only have one non-zero value F'_{i_p}, with the same population size distribution, F'_{i_p} will be much greater than F_{i_p} on average. In this way, the supporting region of the profiles F used in training is increased greatly.

4.2 Feature Engineering and Model Structure

4.2.1 Feature engineering. The raw features include sample profile f and population size N. On top of the two raw features, other meaningful features that we can derive include sample size $n = \sum_i i f_i$, sample NDV $d = \sum_i f_i$, and sampling rate $r = n/N$.

The number of elements in the sample profile f varies for different samples. The length of f can be as large as the sample size and as be as small as one. However, during model training, we have to use a fixed number of features. We choose to keep only the first m elements of the sample profile f, *i.e.* $f[1:m]$. If the length of f is less than m, we pad it with zeros. This is based on the intuition that the predictive power of f_i decreases as i increases. In fact, some of the well-known estimators only use the first few elements in the sample profile f, yet achieving fairly good performance [20, 22, 30]. To make up for the elements being cutoff in the sample profile f, we add the corresponding *cut-off sample size* $n_c = \sum_{i=m+1} i f_i$ and *cut-off sample NDV* $d_c = \sum_{i=m+1} f_i$ as two additional features.

The feature set we use is $x = \{N, n, n_c, d, d_c, 1/r, f_1, \ldots, f_m\}$, with a total of $N_x = m+6$ features, and the single target we aim to predict is population NDV D. We set $m = 100$ by default in experiments.

4.2.2 Model structure. Recall that our goal is to learn an estimator/model to approximate D^{MLE}. Simple models often assume some specific relationship between the input features, *i.e.*, the sample profile f in our case, and the label to be predicted, *i.e.*, NDV D. For example, linear/logistic regression assumes a linear relation between f and (transformed) D: *e.g.*, $D = \sum_i a_i f_i$. However, D^{MLE} can be any *unknown* function of f (*e.g.*, refer to the previous estimator D^{Shlosser} introduced in Section 2.2) that is much more complex than the above restricted class of linear functions. Therefore, since neural networks are able to approximate any function (with enough nodes and layers) [34], we choose our model to be a neural network.

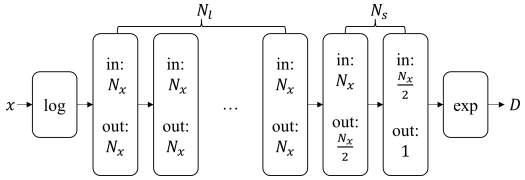

Figure 3: Network architecture: # of linear layers is $N_l + N_s$.

Our network architecture is shown in Figure 3. There are $N_l + N_s$ linear layers in total. The activation function for every layer is LeakyRelu [6]. The first N_l linear layers compose a set of more complex features than the raw features. This is followed by a "summarizer" component with $N_s = 2$ linear layers that gradually summarize features in N_x dimensions to form the final one dimensional prediction. We can control the capacity and complexity of our model by N_l. We set $N_l = 5$ by default and test the sensitivity to N_l in Section 6.4. Since the architecture is very simple, the model inference time is at the scale of microseconds on CPU in our experiments.

There is one additional challenge for model learning: The magnitude of different features in the feature set x can be at different scale. For example, the population size can be as large as 10^7 while the sample NDV d can be as small as 1. This makes it very difficult to learn the model parameters [37, 38]. The common practice to resolve this problem is to perform normalization [37, 38]. For example in z-normalization each feature z is normalized by the mean μ and standard deviation σ of the feature: $z' = \frac{z-\mu}{\sigma}$. The underlying assumption of this practice is that training data and test data are drawn from the same distribution, so that the mean and standard deviation in the test set will be equal to that of the training set. When each test data point comes, we can use the mean and standard deviation in the training set to normalize it. However, in our case, we do not assume the training set and test set to share the same distribution. In fact, our training set is synthetically generated, so it can be very different from the real-world test datasets.

We take the logarithm of features (*e.g.*, sample profile f_i's and inverse sampling rate $1/r$) before the first layer, after adding a small constant to each feature to avoid logarithm of zero. Since the network now operates at log scale, we take exponential on the output of the last linear layer as the final output. Specifically, let $D_{\log}$ denote the output of the last linear layer, and the final estimation is $D = e^{D_{\log}}$. There are three reasons for taking the logarithm of features in the model. First, our learning objective is to reduce the ratio error defined in (6), which is translated into a loss function as the squared difference between $\log D$ and $\log \hat{D}$ in (8). Secondly, taking logarithm makes NDV and different features at the same scale, and thus easier to train the model (otherwise, the training data points with big NDVs dominate the loss). Thirdly, taking logarithm explicitly introduces non-linearity to the model, so it can be more expressive to approximate nonlinear functions, and helps the learned model to generalize better outside the support region of training data [36] to approximate unbounded functions.

4.3 Model Regularization

We now introduce a regularization method tailed for our task by considering negative results on sample-based NDV estimation. Note

that the negative result [22] introduced in Section 2.3 holds for learned estimator as well, and it says that, any learned estimator $\mathsf{h} \in \mathcal{H}$ with sample or sample profile as the feature has a ratio error at least in the order of $\Omega(\sqrt{N/n})$ *in the worst case* (for some dataset), where N is the population size and n is the sample size. Since the sample profile f is an input feature to the learned estimator, we first try to derive an instance-wise negative result by answering the question how large the error could be after observing f.

An instance-wise negative result and its implication. The idea of our instance-wise negative result generalizes the one of a "global" negative result [22]. Consider two columns (multi-sets) $\mathbf{C}_1 = \mathbf{C}_0 \cup \Delta_1$ and $\mathbf{C}_2 = \mathbf{C}_0 \cup \Delta_2$ sharing the common subset $\mathbf{C}_0$, and the difference (Δ_1 v.s. Δ_2) makes their NDVs D_1 and D_2 differ significantly from each other. We draw two samples from $\mathbf{C}_1$ and $\mathbf{C}_2$ with sample profiles f_1 and f_2 observed, respectively. As long as $\mathbf{C}_0$ is large enough, we can show that, with high probability, both the two samples contain values only in $\mathbf{C}_0$, and thus f_1 and f_2 follow the same distribution which are indistinguishable. More formally:

Proposition 1. *For any size-n sample with profile f and NDV d, there exist two size-N columns $\mathbf{C}_1$ and $\mathbf{C}_2$ with NDVs D_1 and D_2, respectively, such that with probability at least γ we cannot distinguish whether the observed f is generated from $\mathbf{C}_1$ or from $\mathbf{C}_2$, with*

$$D_1 = \left\lfloor \frac{N-n}{4n}\left(\ln(\frac{1}{\gamma}) - \frac{2}{e^c}\right)\right\rfloor + d \text{ and } D_2 = d \tag{17}$$

for $\gamma \geq e^{-4n-2e^{-c}}$ and $n \geq d(\ln d + c)$.

Proof of Proposition 1 can be found in our technical report [58]. Proposition 1 suggests that, no matter how well an estimator h is trained under the loss function $\hat{R}_A$ in (9), it is inherently hard to estimate NDV using the model $\mathsf{h}(f, N)$. Consider the two columns $\mathbf{C}_1$ and $\mathbf{C}_2$ with NDVs D_1 and D_2 constructed in Proposition 1. Samples with profiles f_1 and f_2 are then drawn from $\mathbf{C}_1$ and $\mathbf{C}_2$, respectively. Proposition 1 says that, with high probability, f_1 and f_2 follow the same distribution, and thus the expected output of $\mathsf{h}(f_1, N)$ should be the same as the one of $\mathsf{h}(f_2, N)$, considering the randomness in drawing samples. This property of h contradicts to the fact that the true NDVs D_1 and D_2 in (17) differ significantly.

Following the above discussion, we can derive an instance-wise lower bound of the ratio error by forcing $\mathsf{h}(\cdot, N)$ to output $\sqrt{D_1 D_2}$.

Theorem 1. *For any size-n sample with profile f and sample NDV d observed (where $n \geq d(\ln d + c)$), with probability $\gamma \geq e^{-4n-2e^{-c}}$, there exist a choice of column with NDV D such that any estimation of D based on f,* i.e., *$\mathsf{h}(f, N)$, has ratio error at least*

$$\text{error}(\mathsf{h}(f,N), D) \geq \sqrt{\frac{\left\lfloor \frac{N-n}{4n}\left(\ln(\frac{1}{\gamma}) - \frac{2}{e^c}\right)\right\rfloor + d}{d}} \triangleq b(d,n). \tag{18}$$

Proof of Theorem 1 can be found in our technical report [58].

Regularization for a worst-case optimal estimator. When training an estimator h under the loss function $\hat{R}_A$ in (9), consider two training data points $p_1 = ((f, N), D_1)$ and $p_2 = ((f, N), D_2)$ where D_1 and D_2 as in (17) are NDVs of the two columns $\mathbf{C}_1$ and $\mathbf{C}_2$ constructed in Proposition 1. If p_1 is in the training data A but p_2 is not, h tends to predict D_1 after training; if p_2 is in A but p_1 is not, a trained h tends to predict D_2. In both cases, the worst-case ratio error for estimation could be as large as D_1/D_2.

Our goal of regularization here is to push a trained model h towards an *instance-wise worst-case optimal estimator* h^* whose ratio error matches an *instance-wise lower bound* $b(d, n)$ as in (18). To this end, consider a loss function $\hat{R}_{A,\mathsf{h}^*}(\mathsf{h})$ which aims to minimize the distance between h and h^* in prediction:

$$\hat{R}_{A,\mathsf{h}^*}(\mathsf{h}) = \frac{1}{|A|}\sum_{((f,N),D)\in A} |L(\mathsf{h}(f,N),D) - L(\mathsf{h}^*(f,N),D)|. \tag{19}$$

Intuitively, with this loss function in (19), we want our model h to be trained towards the "optimal" estimator h^*. In particular, if $\hat{R}_{A,\mathsf{h}^*}(\mathsf{h}) = 0$, h behaves exactly the same as h^* on the training set A with $L(\mathsf{h}(f,N),D) = L(\mathsf{h}^*(f,N),D)$ for each $((f,N),D) \in A$. Comparing to the loss $\hat{R}_A(\mathsf{h})$ in (9), $\hat{R}_{A,\mathsf{h}^*}(\mathsf{h})$ in (19) allows the model h to have a higher error (only as good as $L(\mathsf{h}^*(f,N),D)$) on the training set, but aims to prevent overfitting.

Loss function with regularization. Recall the loss $L(\cdot, D)$ in (8) used by our model. If h^*'s ratio error matches the *instance-wise lower bound* $b(d, n)$ in (18), we have $L(\mathsf{h}^*(f,N),D) = (\log b(d,n))^2$. We also apply L2 regularization on model parameters W to encourage W to be small and sparse for better generalization [16]. Putting them together, we use the following loss function in training:

$$\hat{R}^*_A(\mathsf{h}) = \frac{1}{|A|}\sum_{((f,N),D)\in A} |L(\mathsf{h}(f,N),D) - (\log b(d,n))^2| + \lambda\|W\|_2. \tag{20}$$

Note that $b(d, n)$ is defined only when $n \geq d(\ln d + c)$ according to Theorem 1. For $n < d(\ln d + c)$ (sample NDV is close to sample size), we define $b(d, n) \triangleq 1$. In our implementation, c is set to be 10 in (18) such that the $2/e^c$ term in $b(d, n)$ is negligible.

There are two hyperparameters here. i) γ for $b(d, n)$ defined in (18) controls how confident this lower bound is. We set $\gamma = 0.6$ by default to have a medium level of confidence. ii) λ controls the strength of the L2 regularization. We set $\lambda = 10^{-1}$ by default in our learned estimator after a tuning process based on training loss. Details about the tuning process can be found in our technical report [58]. It is important to note that, when choosing γ and λ, we use absolutely no knowledge about the test datasets. The robustness of our model to different choices of γ and λ is evaluated in Section 6.4.

5 USAGE AND DEPLOYMENT

We extract the trained weights of our model in "`model_paras.npy`" and implement a numpy version of the model to have minimum dependency on other libraries. We provide the trained model, *i.e.*, our learned NDV estimator in [43]. The usage of it is as simple as a statistical estimator such as GEE and users can easily estimate population NDV by providing either a sample or a sample profile (drawn from a column or query results), as shown below:

```
from estimate import Estimator
estimator = Estimator(para_path="model_paras.npy")
D1 = estimator.sample_predict(S=[1, 1, 1, 3, 5, 5, 9], N=100000)
D2 = estimator.profile_predict(f=[2, 1, 1], N=100000)
```

Our model does not require any re-training for new workloads and the inference time is at the scale of microseconds on CPU as our model structure is very simple. Specifically, in our experiments, inference on a single profile has a running time of 240×10^{-6} seconds. Our trained estimator can be easily plugged into any existing systems. For example, we have plugged our trained model to the cloud big data processing platform MaxCompute [7] at Alibaba.

6 EXPERIMENTS

We conduct experiments to evaluate the efficacy of our proposed method and compare it with baselines along three dimensions:

- *Performance.* How accurate is our learned estimator?
- *Ablation study.* How does our way of generating training data and performing model training contribute to the final performance?
- *Sensitivity analysis.* How sensitive is the performance of our model to different hyperparameters?

6.1 Experiment Setup

Hardware. All our experiments were performed on a machine with a 2.50GHz Intel(R) Xeon(R) Platinum 8163 CPU, a GeForce RTX 2080 Ti GPU and 376GB 2666MHz RAM.

Datasets. We conduct experiments on nine real-world datasets from diverse domains. Note our method does not use any of them for any training or hyperparameter tuning. The datasets are only used for evaluation after our model is trained on synthetic data.

1) *Kasandr* [54, 55]: Behavior records of customers in e-Commerce advertising with 15.8M rows and 7 columns.
2) *Airline* [1, 3]: Summary statistics of airline departures from 1987 to 2013 with 10.0M rows and 10 columns.
3) *DMV* [5, 39]: Data from Department of Motor Vehicles, containing information about cars, their owners and accidents. There are 11.7M rows and 20 columns.
4) *Campaign* [4]: Information of individual contributions to election campaigns. There are 3.3M rows and 21 columns.
5) *SSB* [49]: The star schema benchmark. We use the fact table with a scaling factor of 50, resulting in 300M rows and 17 columns.
6) *NCVR* [12]: North Carolina voter registration data with 8.3M rows and 71 columns.
7) *Product*: Private dataset with information of product items on an online-shopping website. There are 5.2M rows and 25 columns.
8) *Inventory*: Private dataset of inventory statistics. There are 8.8M rows and 19 columns
9) *Logistics*: Private dataset of logistics information of shipping orders. There are 8.6M rows and 28 columns.

Methods evaluated. We compare our method to nine methods:

1) *GEE* [22]: This method is constructed by using geometric mean to balance the two extreme bounds of NDV. It is proved to match a theoretical lower bound of ratio error within a constant factor.
2) *HYBGEE* [22]: This is a hybrid estimator using GEE for high-skew data and using the smoothed jackknife estimator for low-skew data. We refer to Pydistinct [8] to test for high or low skewness.
3) *HYBSKEW* [30]: This is a hybrid estimator using shlosser for high-skew data and using the smoothed jackknife estimator for low-skew data. We refer to the implementation in Pydistinct [8].
4) *AE* [22]: This was proposed to be a more principled version of HYBGEE with smooth transition from low-skew data to high-skew data. This method requires numerically finding the root of a non-linear equation. We use the classic Brent's method [10, 17].
5) *Chao* [20]: This method is derived by approximating the coverage as $1 - f_1/n$ and assuming the population size is infinity. It predicts NDV to be infinite when $f_2 = 0$, so we invoke GEE in this case.
6) *Chao-Lee* [21]: This method adds a correction term to the coverage estimation in Chao to handle skew in data [30].

Table 1: Inference complexity (1st line) and # arithmetic operations (2nd line) needed in implementation

GEE	HYB GEE	HYB SKEW	AE	Chao	Chao -Lee	Shlo sser	APML	Our
O(1)	O(n)	O(n)	-	O(1)	O($\|f\|$)	O($\|f\|$)	O(n)	O(1)
5	$5n$	$5n$	-	4	$4\|f\|$	$7\|f\|+2$	$n+\sqrt{n}\log n$	300 + 51200

7) *Shlosser* [52]: This method is derived based on a skewness assumption: $E(f_i)/E(f_1) \approx F_i/F_1$. The method performs well when each distinct value appear approximately one time on average [30].
8) *APML* [50]: This method analytically approximates the profile maximum likelihood estimation for population profile F^{PML} (solution to (2)), and estimate NDV as $\sum_i F_i^{\text{PML}}$. We use the implementation provided in the original paper. Note this method doesn't assume population size N is given. For a fair comparison, when this method predicts NDV to be greater than N (in this case ratio error can be extremely big), we replace its prediction with the best performing baseline GEE's prediction.
9) *Lower Bound (LB)*: This method is to demonstrate the possible gain of accessing some columns of the real-world test set at training phase. For this method, we split all columns in the the nine real-world datasets by 4:1:5 as training, validation and test set. We fine tune our learned model (trained on synthetic data) on the training set and use the validation set to perform early stopping to prevent over-fitting. We evaluate the fine-tuned model on the remaining test set. This provides an empirical lower bound for the error of any workload agnostic estimators.

Inference cost. For all the methods evaluated, the inference cost, *i.e., processing time of estimating NDV per column*, is usually dominated by the cost of scanning the sample with size n to obtain sample profile f and sample NDV d. *Suppose f and d are already obtained*, Table 1 summarizes the complexity and the number of arithmetic operations needed in the *remaining steps* for inference in different methods. Here, let $|f| = \max\{j \mid f_j > 0\}$ denote the length of f. GEE, Chao, and our method need another O(1) arithmetic operations to derive the NDV estimation, while Chao-Lee and Shlosser need another O($|f|$) arithmetic operations and HYBGEE, HYBSKEW, and APML need another O(n) operations. Our method needs about 300 arithmetic operations for feature engineering and 51200 arithmetic operations for a forward pass in the neural network. AE needs to solve a nonlinear equation numerically.

Setup for various estimators. We evaluate all methods with sampling rates varying from 10^{-4} to 10^{-2}: 10^{-4}, 2×10^{-4}, 5×10^{-4}, 10^{-3}, 2×10^{-3}, 5×10^{-3}, and 10^{-2}. To deal with randomness in e.g. sampling, we run ten times and report the averaged results. We implement our model with pytorch and use Adam [40] as optimizer and perform training with skorch [11]. We implement L2 regularization with its equivalent form - weight decay [13, 56]. For our method, the hyperparameters include: the number of elements in sample profile that are used as features m, the number of linear layers in our network architecture N_l, learning rate during training lr, logarithm of the smallest sampling rate B', logarithm of the largest population size in training data B, number of synthetic training data used N_A, the confidence level γ of our derived lower bound in (18), and the L2 regularization parameter λ. We have explained we set $N_l = 5$ in Section 4.2.2, $B' = -4$ in Section 4.1.2, $B = 9$ in

Section 4.1.1, $\gamma = 0.6$ and $\lambda = 0.1$ in Section 4.3. For the remaining parameters: We set $m = 100$ so the total number of features we use is $N_x = 106$, and we test the sensitivity to m in Section 6.4. We choose a learning rate that has the smallest training loss and also converges in a reasonable time: $lr = 0.0003$. We generate $N_A = 0.72 \times 10^6$ training data points, which takes about two hours running in parallel on our machine and we further vary the amount of training data used in Section 6.4. For sanity, we drop the data points with population size $N < 10^4$ as the sample may contain zero data points because sampling rate can be as small as 10^{-4}.

Performance metric. We used the widely used ratio error as our performance metric, which is defined in (6).

6.2 Performance

Overall performance. The overall performance of all methods is shown in Table 2. Our method achieves the lowest ratio error on seven out of nine datasets and has comparable error to the best baseline methods on the other two datasets. Our averaged ratio error is very close to the empirical lower-bound error. Overall, most baseline methods fail significantly on at least one dataset, which could be caused by the violation of the assumptions they make. GEE is the best performing baseline. Although HYBGEE and AE were both designed to improve over GEE, their performance is very sensitive to choice of dataset, because HYBGEE involves estimating skewness of population which can be difficult on some datasets and AE requires numerically solving a non-linear equation which can be brittle in some cases. APML also doesn't work well. One reason is the analytical approximation in APML can introduce a big error; Another reason is that APML is designed to estimate population profile F and is not tailored for estimating NDV.

Ratio error under different sampling rate. We plot the averaged ratio error under different sampling rate in Figure 4(1). Our method has the lowest error and is close to the empirical lower-bound under all sampling rates. In addition, as sampling rate decreases, the advantage of our method over other methods increases significantly. Shlosser has comparable performance with our method at a high sampling rate 10^{-2}, but its performance decreases rapidly as sampling rate decreases. The error of AE is irregular with respect to sampling rate because error of AE depends on the accuracy of the root finding of a non-linear equation, which is brittle in practice.

Ratio error under different NDV. We plot the averaged ratio error under different number of distinct values in Figure 4(2). Our method has the lowest error under all range of number of distinct values except the extremely small region, *i.e.*, NDV ≈ 1.

Data distributions with high skew (small NDV) and low skew (large NDV) are two relatively "easy" scenarios for NDV estimations. Intuitively, when NDV is extremely small, it can be estimated as NDV in the sample; when NDV is extremely large (close to population size), NDV can be estimated as $\frac{\text{sample NDV}}{\text{sampling rate}}$. For example, as is analyzed in [22] and demonstrated above in Figure 4(2), GEE and Shlosser are known to perform well for data with high skew (NDV < 10); some other estimator, *e.g.*, HYBSKEW and HYBGEE, combines the results of two estimators (one performs well for low skew and the other for high skew)–one of the two is selected depending on a test designed to measure the skew of the data. Therefore, the

Table 2: Ratio error for all methods averaged over all columns in each dataset and over all sampling rates. The error for the empirical lower bound is denoted in grey.

	GEE	HYB GEE	HYB SKEW	AE	Chao	Chao -Lee	Shlo sser	APML	LB	Our
Kasandr	3.4	7.9	8.2	4.6	4.7	9.9	19.2	5.1	2.2	**2.4**
Airline	4.1	1.8	1.8	**1.4**	1.6	1.8	70.0	2.0	2.5	1.9
DMV	5.2	3.5	17.5	7.7	8.8	13.8	23.5	7.6	3.2	**2.7**
Campaign	7.3	6.5	8.4	291.8	13.4	48.0	21.9	13.2	2.9	**3.9**
SSB	5.2	1.2	1.2	**1.1**	**1.1**	1.2	173.1	1.3	1.5	2.0
NCVR	12.2	31.0	56.9	150.2	13.2	58.6	42.6	16.9	5.6	**6.4**
Product	36.3	60.6	58.7	46.1	46.9	250.6	30.5	54.8	9.2	**14.6**
Inventory	17.8	23.5	18.4	75.1	24.8	252.9	11.6	26.7	4.5	**7.8**
Logistics	17.1	93.1	100.0	552.0	15.5	275.1	19.5	16.7	3.6	**3.5**
Average	12.1	25.5	30.1	125.6	14.5	101.3	45.7	16.0	3.9	**5.0**

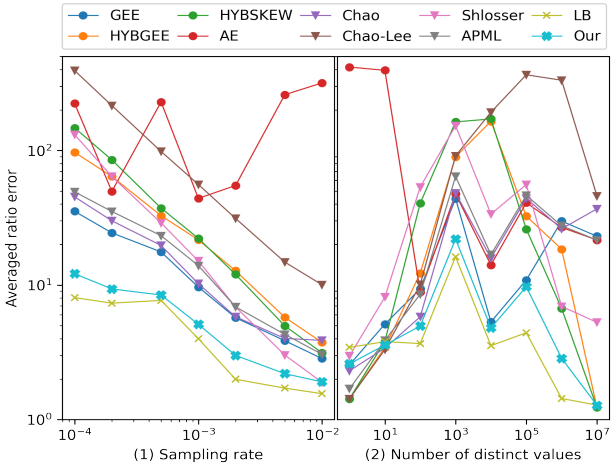

Figure 4: Averaged ratio error over all datasets for different (1) samping rate and (2) NDV.

general trends for most estimators observed in Figure 4(2) are similar: they perform well for small NDV, and as NDV increases, the error first increases and then decreases for large NDVs. Given this general trend, however, the turning points for different methods are different, depending on their concrete forms and data distributions. For example, the turning point of Chao-Lee is 10^5; the turning point of HYBSKEW/HYBGEE is about 10^4, and their performance also largely depends on the skewness test result on the sample data, which may not be stable; the turning point of Shlosser is 10^3. The performance of AE is irregular and largely depends on data distributions, as it considers only estimators of the form $d + Kf_1$ [22], and tries to estimate K from the sample data.

Ratio error distribution. To show the distribution and the worst case of ratio error, we plot the boxplot [2] of ratio error in Figure 5 when sampling rate $r = 10^{-3}$. Overall, our method has the smallest error in most cases and also has the smallest worst case error. In 75% of cases, the ratio error of our method is smaller than 3, much better than all baselines. The worst case ratio error of our method is almost the same as the baseline GEE that has theoretical worst

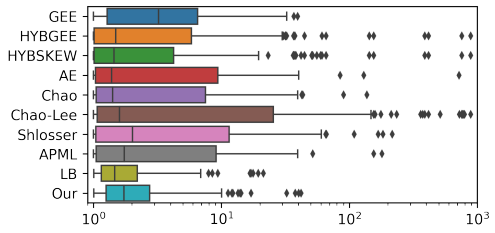

Figure 5: Ratio error on all columns in all datasets when sampling rate $r = 10^{-3}$. In each sub-boxplot for each method, the five vertical lines denote minimum, 25th percentile, median, 75th percentile, and maximum respectively; the colored box contains 50% of the data points; the diamond symbols on the right side of the maximum line are outliers.

case ratio error guarantee. In 75% of cases, the two hybrid methods HYBGEE and HYBSKEW are on average better than GEE but they are not stable and their worst case error can be as large as 1000 causing their averaged ratio error to be greater than GEE in Table 2.

6.3 Ablation Study

We perform ablation study to show the contribution of each component in our method and compare them to existing or straightforward solutions. Table 3 shows the results. Column "Full" denotes our method with full set of components. We ablate our method in three aspects, training data generation, model architecture and objective function for training. The symbol "-" denotes dropping the component that it precedes.

Training data generation. As shown in column "-diversify" in Table 3, when removing the diversify component proposed in Section 4.1.3, ratio error increases a bit, though the error is still significantly lower than the best performing baseline in Table 2. This verifies our intuition in Section 4.1.3: increasing the support region of the profiles is helpful to the generalization ability of our model. We also compare to four heuristic method of generating training data. "uni" denotes generating each F_i of the population profile F by a uniform distribution. "pl" denotes generating F so that the relationship between F_i and i is power law. "rw" denotes generating F by random walk, i.e. $F_{i+1} = F_i + s$ where s is a random step size. "mix" denotes mixing all training data generated by the three heuristic methods. For the three heuristic methods, we also ensure that we introduce enough randomness by using a set of different random hyperparameters (e.g. length of F, range of uniform distribution, parameters in power law distribution, and step size in random walk) for each data point. The results in Table 3 show that our way of generating training data outperforms the heuristic methods significantly. When mixing the three types of heuristically generated data together, the averaged error is smaller than using each type along because mixing all data increases training data diversity and helps the model to generalize better.

Model architecture. We drop the logarithm layer in our model architecture (Figure 3). As shown in column "-log" in Table 3, the ratio error increases dramatically. This validates our choice to take logarithm and its advantages as discussed in Section 4.2.2.

Table 3: Ablation analysis. "-" denotes removing a component. The four columns under "heuristic" replace our training data generation by four heuristic methods respectively.

Datasets	Full	Training data generation						Model arch	Training objective		
		-diversify	heuristic					-log	-b	-b -L2	-b-L2 -log
			uni	rw	pl	mix					
Kasandr	2.4	5.2	40.8	30.8	>1k	50.1		>1k	2.3	3.3	>1k
Airline	1.9	2.5	3.4	4.7	464.0	24.9		>1k	1.8	1.9	>1k
DMV	2.7	3.4	129.8	45.2	14.6	98.3		>1k	2.5	3.8	>1k
Campaign	3.9	6.7	147.9	176.5	933.1	127.9		>1k	3.9	21.6	>1k
SSB	2.0	3.3	19.1	1.2	6.5	3.1		>1k	2.1	1.8	>1k
NCVR	6.4	8.1	83.2	92.9	>1k	113.8		>1k	7.2	8.2	>1k
Product	14.6	20.1	417.6	227.3	52.7	658.5		>1k	20.4	22.0	>1k
Inventory	7.8	10.1	386.7	313.4	>1k	109.5		>1k	9.4	12.0	>1k
Logistics	3.5	9.2	496.7	390.6	>1k	115.5		>1k	3.8	4.8	>1k
Average	5.0	7.7	191.2	142.5	>1k	76.9		>1k	5.9	8.8	>1k

Training objective. Column "-b" in Table 3 denotes removing our proposed regularization for a worst-case optimal estimator in (19). Overall, contrasting the column "Full" to the column "-b", we can see adding the proposed regularization makes the overall ratio error decrease by 0.9, closing the gap to the empirical lower-bound by 45%. This regularization pushes the model to a worst-case optimal estimator, and make it more robust (for datasets where it is not the best, it is close to the best-performing estimator). With this regularization, the ratio error on the most difficult datasets (*e.g.*, Product, and Inventory) decreases significantly more than on other datasets. Removing this regularization does not make the learned estimator "optimal" in specific datasets, but the error may decrease a bit on the easy datasets (*e.g.*, Kasandr, Airline, and DMV). We further drop the L2 regularization (column "-b-L2") and error further increases. This is expected as L2 regularization encourages sparse and small model parameters that generalize better. Finally, we further drop the logarithm on NDV (column "-b-L2-log") to use a naive mean squared loss on NDV and the error increases to be extremely high (similar to the "-log" column under "Model arch").

6.4 Sensitivity Analysis

We show the robustness of our method to the hyperparameters.

Sensitivity to number of features. Our number of features is $N_x = m + 6$ where m is the number of elements in sample profile that used as raw features. We set $m = 100$ by default. To test the sensitivity, we vary m from 10 to 200. As shown in Figure 6(1), as m increases from 100 to 200 (N_x increases from 106 to 206), averaged ratio error doesn't change, so setting $m = 100$ suffices.

Sensitivity to number of layers in network. The number of layers in our network is $N_l + N_s$ where $N_s = 2$. We vary N_l from 0 to 8 so the number of layers varys from 2 to 10. In addtion, we evaluate one extreme case $N_l = 0$ and $N_s = 1$, where the network only has one linear layers with N_x input dimenions and one output demension. The averaged ratio error with respect to the number of layers is shown in Figure 6(2). As the number of layer increases, averaged ratio error decreases due to the increase of model complexity. Although $N_l + N_s = 10$ gives marginally better result, our choice with $N_l = 5$ and $N_l + N_s = 7$ gives good enough result.

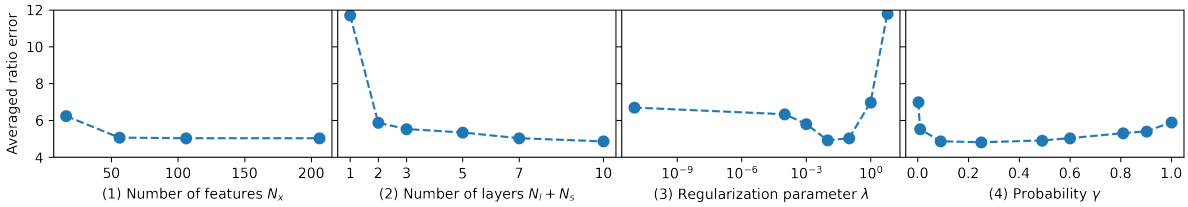

Figure 6: Averaged ratio error vs (1) # of features N_x (2) # of layers $N_l + N_s$ (3) L2 regularization parameter λ (4) Probability γ

Sensitivity to L2 regularization. Figure 6(3) shows the average ratio error under different L2 regularization parameter λ. The error is overall quite stable when λ is in region $[0, 10^{-1}]$. Error increases significantly when λ increases from 10^{-1} to 1 and increases even more dramatically when λ increases from 1 to 10. This increasing trend in region $[10^{-1}, +\infty)$ is similar to the increasing trend of the training loss[58]. Our heuristic method of selecting λ based on the training loss curve at the end of Section 4.3 is able to select $\lambda = 10^{-1}$. Although the optimal λ is at 10^{-2}, $\lambda = 10^{-1}$ is good enough.

Sensitivity to probability γ. Probability γ is the confidence level of our derived lower bound in (18). Figure 6(4) shows the averaged ratio error when probability γ varies in region $(0, 1]$. Overall, the averaged ratio error is quite stable. The optimal value for γ is about 0.2. We heuristically selected γ to be 0.6 in Section 4.3. Although it is not the optimal, $\gamma = 0.6$ is good enough.

Sensitivity to number of training data points. We vary the number of training data points N_A from 10^2 to 7.2×10^6. In Figure 7, the averaged ratio error decreases significantly as N_A increases because more training data improves the generalization ability of the model. However, as N_A further increases, averaged ratio error increases marginally and then stabilizes. This counter-intuitive phenomenon happens often when the distribution of training set is different from the test set. When the training set becomes extremely large, the model learns the very fine-grained details of the training distribution, which tend to not generalize better to the test set.

Choosing training data size. Note that the best choice of training data sizes for different columns can be different. Our goal is to train a workload-agnostic estimator, and it can be too expensive (if not impossible) to tune the training data size for each different workload. From Figure 7, which reports the average error across a number of tables for varying N_A, a general trend is that: after N_A exceeds some threshold (e.g., 10^4), the performance of the learned estimator stabilizes. Thus, in practice, we simply choose a training data size that is large enough (exceeding the threshold) but does not make the training data preparation and model training too expensive. We set the training data size $N_A = 0.72 \times 10^6$ to train the estimator (available in [43]) used in all the other experiments.

Out-of-range generalization. In our training set, the max population size is 10^9 and the max population NDV is 10^9. To evaluate the out-of-range performance, *i.e.*, *estimation for a column with large NDV not seen at training time*, of our learned estimator, we create such extended test datasets: for each column in the original test datasets, we duplicate it 10^4 times and add a suffix i to each value in the ith duplicate. In this way, columns in the extended datasets have extremely large population sizes (maximum at 3×10^{12}) and NDVs

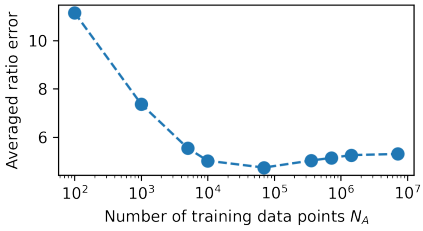

Figure 7: Averaged ratio error vs number of training data N_A.

Table 4: Error on columns with out-of-range NDV ($> 10^{10}$)

	GEE	HYB GEE	HYB SKEW	AE	Chao	Chao -Lee	Shlo sser	APML	Our
Avg ratio error	20.6	30.1	78.3	15.6	143.2	164.5	15.7	37.5	**4.2**

(maximum at 0.9×10^{11}). We focus on columns with population size in $[10^{10}, 3 \times 10^{12}]$ and population NDV in $[10^{10}, 0.9 \times 10^{11}]$, and report the average ratio error for all methods with sampling rate 10^{-3} on these columns in Table 4. Note that both the original and extended test datasets are *not* used for training our estimator. It can be seen that our method also works very well even when NDV to be estimated is out-of-range, and is still better than other baselines. The out-of-range generalizability of our learned estimator is partly due to the logarithm operations as discussed in Section 4.2.2.

7 CONCLUSION AND FUTURE WORK

In this paper, we consider a fundamental question: *whether it is possible to train a workload-agnostic machine learning model to approximate principled statistical estimators such as maximum likelihood estimators (MLE)*. We provide a positive answer to this question on the concrete task of estimating the number of distinct values (NDV) of a population from a small sample. We formulate the sample-based NDV estimation problem as an MLE problem which, however, is difficult to be solved even approximately. We propose a learning-to-estimate framework to train a workload-agnostic model to approximate the MLE estimator. Extensive experiments on nine datasets from diverse domains demonstrate that our learned estimator is robust and outperforms all baselines significantly.

For future work, we would like to extend our learning-to-estimate method to learn estimators for other properties whose MLE is difficult to obtain, *e.g.*, entropy and distance to uniformity.

REFERENCES

[1] 2020. Airlines Departure Delay. https://www.openml.org/d/42728

[2] 2020. Box plot. https://en.wikipedia.org/wiki/Box_plot

[3] 2020. Bureau of Transportation Statistics. https://www.transtats.bts.gov/

[4] 2020. Campaign finance data. https://www.fec.gov/data/

[5] 2020. Department of Motor Vehicle (DMV) Office Locations. https://catalog.data.gov/dataset/department-of-motor-vehicle-dmv-office-locations

[6] 2020. Leaky ReLU. https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html

[7] 2020. MaxCompute. https://www.alibabacloud.com/product/maxcompute

[8] 2020. Pydistinct - Population Distinct Value Estimators. https://pydistinct.readthedocs.io/

[9] 2020. Random numbers that add to 100: Matlab. https://stackoverflow.com/questions/8064629/random-numbers-that-add-to-100-matlab

[10] 2020. scipy.optimize.brentq. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.brentq.html

[11] 2020. skorch documentation. https://skorch.readthedocs.io/en/stable/

[12] 2020. Voter Registration Statistics. https://www.ncsbe.gov/results-data/voter-registration-data

[13] 2020. weight decay in neural networks. https://metacademy.org/graphs/concepts/weight_decay_neural_networks

[14] 2021. Random Vectors with Fixed Sum - File Exchange - MATLAB Central. https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum [Online; accessed 27. Apr. 2021].

[15] Christos Anagnostopoulos and Peter Triantafillou. 2015. Learning to accurately count with query-driven predictive analytics. In *2015 IEEE international conference on big data (big data)*. IEEE, 14–23.

[16] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

[17] Richard P Brent. 1973. Algorithms for Minimization without Derivatives, chap. 4.

[18] John Bunge and Michael Fitzpatrick. 1993. Estimating the number of species: a review. *J. Amer. Statist. Assoc.* 88, 421 (1993), 364–373.

[19] Raymond L Chambers, David G Steel, Suojin Wang, and Alan Welsh. 2012. *Maximum likelihood estimation for sample surveys*. CRC Press.

[20] Anne Chao. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of statistics* (1984), 265–270.

[21] Anne Chao and Shen-Ming Lee. 1992. Estimating the number of classes via sample coverage. *Journal of the American statistical Association* 87, 417 (1992), 210–217.

[22] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 2000. Towards estimation error guarantees for distinct values. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 268–279.

[23] Moses Charikar, Kirankumar Shiragur, and Aaron Sidford. 2019. Efficient profile maximum likelihood for universal symmetric property estimation. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 780–791.

[24] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *SIGMOD*. 511–519.

[25] Reuven Cohen and Yuval Nezri. 2019. Cardinality Estimation in a Virtualized Network Device Using Online Machine Learning. *IEEE/ACM Transactions on Networking* 27, 5 (2019), 2098–2110.

[26] Anshuman Dutt, Chi Wang, Vivek R. Narasayya, and Surajit Chaudhuri. 2020. Efficiently Approximating Selectivity Functions using Low Overhead Regression Models. *Proc. VLDB Endow.* 13, 11 (2020), 2215–2228.

[27] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity estimation for range predicates using lightweight models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057.

[28] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the Analysis of Algorithms Conference*. 137–156.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2017. *Deep learning Ch. 5 Machine Learning Basics*. The MIT Press, 132–133.

[30] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Lynne Stokes. 1995. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB*, Vol. 95. 311–322.

[31] Peter J Haas and Lynne Stokes. 1998. Estimating the number of classes in a finite population. *J. Amer. Statist. Assoc.* 93, 444 (1998), 1475–1487.

[32] Yi Hao and Alon Orlitsky. 2019. The broad optimality of profile maximum likelihood. In *Advances in Neural Information Processing Systems*. 10991–11003.

[33] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment* 11, 4 (2017), 499–512.

[34] Simon Haykin. 1998. *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

[35] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: learn from data, not from queries! *Proceedings of the VLDB Endowment* 13, 7 (2020), 992–1005.

[36] J Wesley Hines. 1996. A logarithmic neural network architecture for unbounded non-linear function approximation. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, Vol. 2. IEEE, 1245–1250.

[37] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (Lille, France) *(ICML'15)*. JMLR.org, 448–456.

[38] Piotr Juszczak, D Tax, and Robert PW Duin. [n.d.]. Feature scaling in support vector data description. Citeseer.

[39] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating join selectivities using bandwidth-optimized kernel density models. *Proceedings of the VLDB Endowment* 10, 13 (2017), 2085–2096.

[40] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[41] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677* (2018).

[42] Seetha Lakshmi and Shaoyu Zhou. 1998. Selectivity estimation in extensible databases-a neural network approach. In *VLDB*, Vol. 98. 24–27.

[43] Library. 2021. An learned sample-based NDV estimator. https://github.com/wurenzhi/learned_ndv_estimator. [Online; accessed 11-October-2021].

[44] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*. 53–59.

[45] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2008. Why go logarithmic if we can go linear? Towards effective distinct counting of search traffic. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. 618–629.

[46] Hamid Mohamadi, Hamza Khan, and Inanc Birol. 2017. ntCard: a streaming algorithm for cardinality estimation in genomics data. *Bioinformatics* 33, 9 (2017), 1324–1330.

[47] Rajeev Motwani and Sergei Vassilvitskii. 2006. Distinct values estimators for power law distributions. In *2006 Proceedings of the Third Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM, 230–237.

[48] Suman Nath, Phillip B Gibbons, Srinivasan Seshan, and Zachary Anderson. 2008. Synopsis diffusion for robust aggregation in sensor networks. *ACM Transactions on Sensor Networks (TOSN)* 4, 2 (2008), 1–40.

[49] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. 2007. The star schema benchmark (SSB).

[50] Dmitri S Pavlichin, Jiantao Jiao, and Tsachy Weissman. 2019. Approximate Profile Maximum Likelihood. *Journal of Machine Learning Research* 20, 122 (2019), 1–55. http://jmlr.org/papers/v20/18-075.html

[51] Maithra Raghu, Ben Poole, Jon M. Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. 2017. On the Expressive Power of Deep Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*. 2847–2854.

[52] A Shlosser. 1981. On estimation of the size of the dictionary of a long text on the basis of a sample. *Engineering Cybernetics* 19, 1 (1981), 97–102.

[53] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 60.

[54] Sumit Sidana, Charlotte Laclau, Massih R Amini, Gilles Vandelle, and André Bois-Crettez. 2017. KASANDR: a large-scale dataset with implicit feedback for recommendation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1245–1248.

[55] Daniel Ting. 2019. Approximate Distinct Counts for Billions of Datasets. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 69–86. https://doi.org/10.1145/3299869.3319897

[56] Twan Van Laarhoven. 2017. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350* (2017).

[57] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready for Learned Cardinality Estimation? *Proc. VLDB Endow.* 14, 9 (May 2021), 1640–1654. https://doi.org/10.14778/3461535.3461552

[58] Renzhi Wu, Bolin Ding, Xu Chu, Zhewei Wei, Xiening Dai, Tao Guan, and Jingren Zhou. 2021. An learned sample-based NDV estimator (technical report). https://figshare.com/s/8cd5f3dad9418b84b75a. [Online; accessed 11-October-2021].

[59] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2021. How neural networks extrapolate: From feedforward to graph neural networks. In *ICLR*.

[60] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proc. VLDB Endow.* 14, 9 (May 2021), 1489–1502. https://doi.org/10.14778/3461535.3461539