Distinct Value Estimation from a Sample: Statistical Methods vs. Machine Learning

For SIGMOD'22 reviewers only.

Authors
authors@email

ABSTRACT

Estimating the number of distinct values (NDV) in a dataset is an important operation in modern database systems for many tasks, including query optimization. In large scale systems, tables often contain billions of rows and wrong optimizer decisions can cause severe deterioration in query performance. Additionally in many situations, such as having large tables or NDV estimation after the application of filters, it is not feasible to scan the entire dataset to compute the number of distinct values. In such cases, the only available option is to use a dataset sample to estimate the NDV. This, however, is not trivial as data properties of the sample usually do not mirror the properties of the full dataset. Approaches in related work have shown that this kind of estimation is connected to large errors. In this paper, we present two novel approaches for the problem of estimating the number of distinct values from a dataset sample. Our first approach presents a novel statistical estimator that shows good and robust results across a broad range of datasets. The second approach is based on Machine Learning (ML), hence being the first time that ML is applied to this problem. Both approaches outperform the state-of-the-art, with the ML approach reducing the average error by 3x for real-world datasets. Beyond pure prediction quality, both our approaches have their own set of advantages and disadvantages, and we show that the right approach actually depends on the specific application scenario.

CCS CONCEPTS

• **Information systems → Data management systems**.

KEYWORDS

NDV, Distinct Values, Query Optimization, Machine Learning

ACM Reference Format:
Authors. 2022. Distinct Value Estimation from a Sample: Statistical Methods vs. Machine Learning. In *SIGMOD '22: International Conference on Management of Data, June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/1122445.1122456

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Estimating the number of distinct values of a dataset or table attribute is an important operation in modern database systems, which is also known as the approximate count distinct problem. This paper investigates this problem with the additional constraint that it is not possible to scan the whole dataset for estimation.

The general problem can be described as follows: we are analyzing a *multiset*[1] with a total population size of N elements. Each element of the multiset has a *'key'* value, with each key potentially having a different *frequency*: the number of times it occurs in the multiset. We are allowed to take a sample of r elements from this multiset (or dataset)[2]. Based on this sample, the goal is to predict the number of distinct keys in the multiset (also known as the number of distinct values or NDV). A simple example of a multiset in databases is a table column. The population size is equal to the number of rows in the table and the number of unique keys in the column (NDV) is the target to be estimated. For example, in a *'months'* column, the NDV value is likely to be 12.

NDV estimates are essential for many database operations. For example, the query optimizer may rely on them to estimate result sizes of join or group-by operators [18]. The estimates can be used to determine the join order or to optimize the succeeding query operators, resulting in better and more robust query plans. Another application of NDV estimates is resource allocation for indexes or hash tables. The former is important to decide if an index is worth its memory overhead (e.g., for auto-index-creation [8]), while the latter is important for performance reasons. Filling a hash table with a bucket size that is too small for the number of keys, results in orders of magnitude higher access latency due to chaining or rehashing [1, 17].

The naïve approach to calculate the NDV is scanning the full dataset and calculating the exact number of distinct values. Counting the exact number is resource intensive as either large intermediate structures need to be maintained (e.g., a hash table) or the data needs to be scanned multiple times. Many *one-pass* approaches [10, 12, 16, 34] have been proposed to address this problem that scan the whole table but reduce the cost of intermediate structures. However, scanning the entire data is often not feasible due to time constraints for datasets with billions of elements, especially in systems where the data can not be kept in main memory. Therefore, the only practical solution is to collect a random sample of the dataset and to estimate the NDV based on this sample.

Estimating the number of distinct values from a sample, however, is a challenging problem, where even the most accurate estimators occasionally produce large errors. In addition, there are strong

[1] A multiset is a modification of the concept of a mathematical set, that allows each of its elements (or keys) to occur multiple times.

[2] In this paper the terms multiset and dataset are used interchangeably.

negative results [5, 6] showing that no estimator can guarantee good results against an adversarial choice of input data. However, while the inability to provide guaranteed robust bounds is unfortunate, there are a large number of database scenarios, where there is no practical alternative to estimating NDV from a sample. Despite the need for sampling based estimation for large datasets, there has not been much recent research on the problem, and most commercial products are tending towards one-pass approaches.

To show the feasibility of NDV estimation based on sampling, we present *two* novel and inherently different approaches in this paper. The two approaches are (1) a new statistical estimator based on a Binomial model of key selection [5, 33], and (2) a Machine Learning (ML) approach, build upon an ensemble-based regression model, while encoding extreme parts of the problem as an additional classification task. To the best of our knowledge, this is the first Machine Learning approach to address this problem. We use a broad range of datasets and standard metrics for our evaluation, while introducing a new metric, called *singed relative error (sRE)*, to better understand the predictions. Both approaches significantly exceed the prediction quality compared to current state-of-the-art estimators, while each approach has its own advantages and disadvantages. In fact, these two approaches were designed and developed independently for two different application environments because of their specific properties. Therefore, we not only evaluate their prediction quality in this paper, but also discuss their properties and applicability to various application scenarios.

The paper is structured as follows. We discuss related work in Section 2, introduce our statistical estimation approach in Section 3, and present our ML approach in Section 4. Afterwards, we evaluate both approaches in terms of accuracy in Section 5 and compare the approaches along other attributes in Section 6. Finally, we conclude the paper in Section 7.

2 RELATED WORK

Calculating the number of distinct keys in a multiset on-demand incurs two major costs: (1) the cost of scanning the entire multiset and (2) the cost of maintaining an in-memory structure (such as a hash table), to store the keys already observed during the scan. There is a large body of work on *one-pass* count distinct estimation [10, 12, 16, 34] with the goal of scanning the full dataset once, while keeping a light-weight in-memory structure that stores an estimate NDV. While this is a significant improvement in memory space, it does not reduce the scanning cost. In the context of database systems, in situations where it is not possible to scan the entire table, database structures like histograms or indexes can be used to estimate the NDV [31, 32]. However, estimates from these structures often need to be modified (e.g., when query filters are applied), which can introduce large errors. In addition, there are many situations, where maintaining a histogram or index may not be feasible or might be too expensive. In such situations, the NDV can only be estimated using a sample.

This problem is not unique to database systems as for example file duplication in storage systems [36] or estimating animal populations [3, 4] experience the same challenge. A large number of statistical techniques have been developed to address the problem of estimating the number of distinct values from a sample [9, 14, 15]. Several studies [9, 15] provide a comparison of the accuracy of these techniques. Deolalikar et al. [9] provide a more recent survey, specifically focusing on Zipfian distributions [38], due to their growing prominence in large datasets. Though all techniques show significant error, they identify the Adaptive Estimator (*AE*) [5] as having relatively better performance, compared to other methods.

AE is a model-based estimator that, along with the Shlosser estimator [33], can be considered to belong to a family of estimators that share a common Bernoulli sampling based model of key selection. The Shlosser [33] estimator differs from *AE* in explicitly assuming that the key frequencies follow a Zipf distribution. This makes the Shlosser estimator highly effective for Zipf distributions, but inaccurate for other distributions. To address this, Haas et al. [15] proposed a hybrid approach combining the Shlosser estimator [33] and the smoothed Jacknife estimator [4], while switching between them depending on data properties.

This paper introduces a new estimator, the Histogram Normalization Estimator (*HNE*), which uses the Bernoulli sampling model of key selection, but corrects for sampling errors that can cause estimators such as AE to provide highly inaccurate NDV estimates. We show that as a result, HNE outperforms other Bernoulli estimators across a range of datasets. In addition to *HNE*, this paper presents a machine learning based approach to NDV prediction, completely different to existing statistical methods.

3 HISTOGRAM NORMALIZATION ESTIMATOR

This section first provides a description of the estimation problem and introduces the notation. Following this, we briefly describe the Binomial family [5, 33] of NDV estimators. Then we provide a derivation for the *histogram normalization estimator* (HNE), a novel count distinct estimator.

3.1 Problem Formulation and Binomial Model

The problem input is a sample from a dataset of population size N, with an unknown number of distinct keys D, characterized by: (1) the set of distinct keys observed in the sample, and (2) the number of times each of these keys is observed in the sample. Let f_i be the number of keys in the sample that were observed i times (so, for example, f_2 represents the number of keys included twice in the sample). Then the sample is represented by the vector $\boldsymbol{f} = (f_0, f_1, \ldots, f_r), 0 \leq f_i \leq r$, where r is the sample size. While $f_1, \ldots, f_r$ are observed, f_0 is unknown. A large number of values in $\boldsymbol{f}$ could be zero. The number of distinct keys observed in the sample can then be written as $d = \sum_{i=1}^{r} f_i$. Since $D = f_0 + d = \sum_{i=0}^{r} f_i$, our goal is to estimate f_0. We call these f_0 keys the *missing keys*, as these keys are missing from the sample. To avoid confusion, we call the number of times a key is observed in the sample the *size of the key* in the sample, while we call the number of times a key of a certain size is present in the sample, the *frequency of the key size* in the sample. So, for example, f_2 represents the frequency with which keys of size 2 are present in the sample. Similarly, we call the number of times a key is present in the dataset, the size of the key in the dataset.

The Binomial model of key selection [5, 33] models each key in the sample as drawn from a Bernoulli process with r draws

(r being the sample size). Thus the probability that a key with size S in the dataset is observed i times in the sample is given by the Binomial probability $Pr(i; r, \frac{S}{N}) = \text{Bin}\left(k = i, n = r, p = \frac{S}{N}\right)$. Under this model, f_0 is the number of Binomial experiments, out of a total of D, that yielded 0 successes. A derivation of the model from a hyper-geometric model of key selection is provided by Shlosser et al. [33].

The Adaptive Estimator (AE) [5] uses the Binomial model and models the sample as drawn from k components (sub-populations), one for each key size i having $f_i > 0$. All keys in the i^{th} sub-population have a Binomial probability $Pr\left(0; r, \frac{i}{r}\right)$ of being absent (or missed) in the sample. This probability is higher for what we call the *small key* components, that is, sub-populations corresponding to keys observed once or twice in the sample. AE models the relative frequency of keys of size i in the dataset as proportional to f_i. However, this approximation is vulnerable to sampling error as, for example, a significant portion of the keys observed twice in the sample could possibly have a probability $\frac{i}{r}$ of selection, where $i \geq 3$. These sampling errors can have an outsized impact because AE uses a separate model for keys missing from components $i \in \{1, 2\}$.

Section 3.3.1, presents a method we call *histogram normalization* to correct for such sampling errors in key frequency estimation. As AE cannot be easily modified to incorporate these corrections, we develop an alternative estimator which can include them, in the next section (Section 3.2). This estimator is then modified to incorporate both: a) histogram normalization (Section 3.3.1), and also, b) a more elaborate model for estimating the number of missing *small keys* (Section 3.3). We refer to the resulting estimator as the *Histogram Normalization Estimator* (HNE). Following this, Section 3.3.2 discusses HNE derived bounds that can be provided on NDV estimates.

3.2 Naïve NDV Estimator

This section provides a naïve estimator for an upper bound on $E[f_0]$, the expected number of missing keys in the sample. We consider it a naïve estimator because: a) it assumes that a key's size in the sample is an adequate estimate of its Bernoulli selection probability, and b) it does not correct for sampling errors. These drawbacks are corrected in Sections 3.3 and 3.3.1, leading to the Histogram Normalization estimator (HNE). The naïve estimator is based on the following assumption:

Assumption A1: Let $S = \{i, f_i \in \boldsymbol{f}, i > 0, f_i > 0\}$, be the set (of size k), consisting of key sizes for which non-zero frequencies were observed in the sample. Then the sample $\boldsymbol{f}$ is assumed to be generated by a mixture of k Binomial distributions, $\{Pr(j; r, \frac{i}{r}), i \in S\}$.

Proposition 3.1. *Given Assumption A1, an upper bound estimate $\hat{E}[f_0]$ on the expected number of missing keys f_0 in the sample is given by:*

$$\hat{E}[f_0] \leq \sum_{i \in S} \frac{Bin\left(k = 0, n = r, p = \frac{i}{r}\right)}{Bin\left(k = i, n = r, p = \frac{i}{r}\right)} \cdot f_i \tag{1}$$

Proof. Let f_0^i be the number of keys belonging to mixture component $i \in S$, that were not observed in the sample. Then:

$$f_0 = \sum_{i \in S} f_0^i \tag{2}$$

$$\Rightarrow E[f_0] = \sum_{i \in S} E\left[f_0^i\right] \tag{3}$$

Let $\boldsymbol{F} = (F_1, F_2, \ldots, F_k)$ be a vector, such that F_i is the (unknown) number of keys belonging to the i^{th} component in S. And let $P_{(r+1)\times k}$ be a matrix, such that $P_{ji}, 0 \leq j \leq r, 1 \leq i \leq k$ is the Binomial probability of sampling a key j times from the i^{th} mixture component. Then:

$$E[\boldsymbol{f}] = P \cdot \boldsymbol{F} \tag{4}$$

As a special case of eq. (4):

$$E[f_0^i] = P_{0i} F_i \tag{5}$$

Since the linear system in eq. (4) may not have a solution with positive F_i, we instead use an upper-bound on F_i. Ignoring non-diagonal values for each row in eq. (4), for each f_i:

$$E[f_i] \geq P_{ii} F_i \tag{6}$$

$$\Rightarrow F_i \leq \frac{E[f_i]}{P_{ii}} \tag{7}$$

Replacing eq. (7) in eq. (5):

$$E\left[f_0^i\right] \leq \frac{P_{0i}}{P_{ii}} E[f_i] \tag{8}$$

In order to get an estimate $\hat{E}[f_0]$, we treat the observed values of f_i in $\boldsymbol{f}$ as the expected values $E[f_i]$. We observe that, once we replace the expected values $E[f_i]$ with the observed values f_i, eq. (8) is only non-zero for $i \in S$. Based on this, replacing eq. (8) in eq. (3) and expanding the terms gives:

$$\hat{E}[f_0] \leq \sum_{i \in S} \frac{\text{Bin}\left(k = 0, n = r, p = \frac{i}{r}\right)}{\text{Bin}\left(k = i, n = r, p = \frac{i}{r}\right)} \cdot f_i \tag{9}$$

□

We made the following simplifying assumption in the above proof: that each of the unobserved keys has exactly the same probability of selection as the Bernoulli probability of selection of one of the sampled keys. Given the limited information inherent in a sample, and the large set of possible sizes a key can have in the dataset, this seems reasonable. Note that other statistical estimators [5, 33] make similar simplifying assumptions.

However, the above estimator is not accurate for keys observed once or twice in the sample. This is because they are often a subset of a larger group of keys, each with a much lower individual probability of selection (than $\frac{1}{r}$ or $\frac{2}{r}$), a fraction of which were randomly selected into the sample from amongst their peers. For example, they might be keys occurring once in the dataset (*singletons*), and a random subset of singletons were selected into the sample. Due to this reason, we use eq. (1) to only calculate the values $E[f_0^3]$ and above. The final estimate is then the sum of the following three values, each estimated separately:

- *Missing Large Keys:* Estimated as $\hat{E}\left[f_0^{3\ldots N}\right] \sum_{i=3}^{r} f_0^i, i \in S$, the estimated number of missing keys from component sub-populations with $i \geq 3$.

- *Observed Large Keys*: The observed (non-missing) number of large keys in the sample ($f_{\text{obs}} = \sum_{i=3}^{N} f_i$).
- *Small Keys:* The estimated number of small sized keys (keys similar to those with observed frequency 1 or 2 in the sample, written as m.

Then the final estimate of the dataset NDV is given by:

$$D_{est} = \hat{E}\left[f_0^{3\ldots N}\right] + f_{\text{obs}} + m \tag{10}$$

The naïve estimator thus allows us to divide the estimation problem into sub-problems, which was not possible with AE. This division allows us to address the estimation of m as a separate problem, that can be solved analytically (in the next section), and also allows us to correct for sampling errors when estimating m (Section 3.3.1).

3.3 Missing Small Keys Estimation

To estimate the number of missing small keys, we make the same assumption as AE: that all the small keys are drawn from a single component, where all keys in the component have the same size. Let m be the total number of small keys in the dataset, all having the same size. We can write the number of missing small keys in two ways: (1) as $m - f_1 - f_2$ (since f_1 and f_2 is the subset of the small keys that were observed), and (2) using a Binomial model with m as a parameter. Equating the two, we solve for m. In more detail, let m be the total number of small keys. The number of rows in the sample that consist of small keys is estimated as $r_s = f_1 + 2f_2$. As we assume that all the small keys have the same size, their Bernoulli probability p_s of selection in the sample is given by:

$$p_s = \frac{r_s}{rm} = \frac{f_1 + 2f_2}{rm} \tag{11}$$

Let f_0^s represent the number of small keys that are missing from the sample. Then:

$$E[f_0^s] = \text{Bin}\left(k = 0, n = r, p = \frac{r_s}{rm}\right) \cdot m \tag{12}$$

Also, since:

$$E\left[f_1\right] = \text{Bin}\left(k = 1, n = r, p = \frac{r_s}{rm}\right) \cdot m \tag{13}$$

Using the observed value f_1 for $E\left[f_1\right]$ in eq. (13), we get an estimate $\hat{m}$ of m as:

$$\hat{m} = \frac{f_1}{\text{Bin}\left(k = 1, n = r, p = \frac{r_s}{rm}\right)} \tag{14}$$

Using this value of $\hat{m}$ for m in eq. (12), expanding the Binomial expressions, and simplifying:

$$E\left[f_0^s\right] = \frac{m}{r_s}\left(1 - \frac{r_s}{rm}\right) \cdot f_1 \tag{15}$$

Also, since $E[f_0^s] = m - E[f_1] - E[f_2]$, we equate this to eq. (15), and use the observed values of f_1 and f_2 as the expected value. Then solving for m gives:

$$m = \frac{f_1 + 2f_2}{2f_2}\left(f_1\left(1 - \frac{1}{r}\right) + f_2\right) \tag{16}$$

Since eq. (16) is not defined for $f_2 = 0$, we set $m = f_1 \cdot \frac{N}{r}$ if $f_2 = 0$ (unique keys).

3.3.1 *Histogram Normalization*.

In the previous derivation, we assumed that all keys with size 1 or 2 in the sample, are not drawn from the larger sub-populations. However, in reality, a significant portion of the observed small keys could be large keys that were sampled once or twice by accident. This is an important problem because, as eq. (16) shows, the final estimate of NDV is highly dependent on f_1 and f_2. To address this, we apply a correction to our estimates of f_1 and f_2. The corrected estimates of f_1 and f_2, written as f_1' and f_2' respectively, are estimates of what subset of these keys observed once are actually small keys, and not under-sampled large keys. The correction is made by expanding the f_1 term in eq. (4) (after ignoring the zero-valued F_i terms):

$$f_1 = \sum_{i \in S} P_{1i} F_i = f_1' + \sum_{\substack{i \in S \\ i > 2}} P_{1i} F_i \tag{17}$$

Using the upper-bound in eq. (7) gives:

$$f_1' \geq f_1 - \sum_{\substack{i \in S \\ i > 2}} \frac{P_{1i}}{P_{ii}} f_i \tag{18}$$

Similarly:

$$f_2' \geq f_2 - \sum_{\substack{i \in S \\ i > 2}} \frac{P_{2i}}{P_{ii}} f_i \tag{19}$$

We use the values of f_1' and f_2' calculated above to calculate m using eq. (16). While technically f_1' and f_2' over-correct for large keys, empirically we find in our experiments that using these corrected values significantly outperforms using the original values f_1 and f_2. We do use one heuristic to guard against over-correction: we do not rely on f_1' and f_2' estimates if f_1' is set to 0 or f_2' is set to ≤ 1. In such cases, we first recalculate f_1' and f_2', after setting $i > 3$. If f_1' is still equal to 0, or $f_2' \leq 1$, we use the original values of f_1 and f_2.

3.3.2 *NDV Upper Bound using HNE*.

The *HNE* estimator assumes that all small keys (Section 3.3) have the same size. This condition can be relaxed by assuming that only a subset g of f_1' keys have the same size as the f_2' keys, while $f_1' - g$ of the f_1' keys have a fixed size t. Then the small key NDV can be estimated as:

$$m_g = (f_1' - g)\,\frac{N}{rt} + \frac{g + 2f_2'}{2f_2'}\left(g\left(1 - \frac{1}{r}\right) + f_2'\right) \tag{20}$$

It may be possible to set appropriate values for g and t, using information such as database table statistics. In the absence of such information, it can be seen that setting $t = 1, g = 0$ maximizes m_g. The NDV upper bound D_{UB} is then given by:

$$D_{UB} = \frac{N}{r} f_1' + f_2' + \hat{E}\left[f_0^{3\ldots N}\right] \tag{21}$$

This corresponds to the case where there are a large number of keys with frequency 1, or *singletons* in the dataset. This D_{UB} estimate is similar (though not identical) to the worst-case upper bound established in [6], and is often too high to be useful.

However, database systems often require an upper bound on the NDV estimate, even if there is a risk of occasional underestimation. To address this, we find that empirically, the geometric mean of the *HNE* estimate and the NDV upper bound is able to provide a worst case upper bound on the ground truth NDV, even for highly skewed datasets. Intuitively, the use of geometric mean can be understood as follows: in situations where we suspect that the dataset might contain singletons, we might prefer to be as close to the HNE estimate as possible, while trying to minimize the error due to the presence of singletons. A common error measure used in NDV literature is the *error ratio*, defined as:

$$\text{Error Ratio} = \max\left(\frac{\text{True NDV}}{\text{Estimate}}, \frac{\text{Estimate}}{\text{True NDV}}\right) \tag{22}$$

To minimize our error with respect to this error measure, we could use the geometric mean (rounded to the closest integer) as an upper bound on the NDV ($D_{\text{gm}} = \sqrt{D_{est} \cdot D_{UB}}$).

4 MACHINE LEARNING APPROACH

In the previous section, we introduced a new statistical estimator. In this section, we present a novel approach using ML to predict the NDV of a dataset given only a data sample.

Traditional approaches to the NDV problem [5, 15] use manually-tuned statistical methods. This manual tuning involves adjusting and extending the theoretical principles of the model itself, which needs an expert in the field to do so.

Classical ML algorithms, on the other hand, are usually out-of-the-shelf tools provided by different libraries in nearly all programming languages. The main complexity for an ML approach is to find the right set of features, choose a model, and fine-tune the models hyper-parameters. For the later two points, there is already a selection of AutoML tools, which can do this automatically [7, 11, 37]. With the ML approach, our goal is to utilize this already existing ML environment, to simplify and improve the NDV prediction.

In the following, we introduce the key ideas as our general ML approach, a regression model, to predict the NDV of a dataset. Afterwards, we present optimizations to the initial model, which let the model perform better for cases with NDVs close to zero or close to the population size. Finally, we give an overview on how model training and inference is performed.

4.1 General ML Approach

For the general approach, we devise a single regression model that predicts the NDV of a dataset. As stated before, choosing and engineering the right input features is the most important part for the model. In this section, we show the key ideas for defining the input features and target data (label) to train our regression model.

4.1.1 ***Using Multiple Samples.***
Approaches in the literature usually use a random data sample of a certain percentage and estimate the NDV based on the properties seen in this sample.

One of our key ideas for the ML model is taking multiple samples and comparing the unique keys in these different samples. This can also be achieved by taking only one sample and dividing it randomly into multiple *sample chunks*.

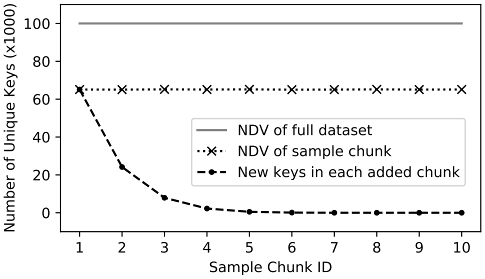


Figure 1: Example of a dataset with 100K unique keys, where each key occurs exactly 10 times. Samples are taken randomly in chunks of 100K without replacement.

Figure 1 shows an example of gaining information by using multiple sample chunks. The dataset contains 100K unique keys, while each key occurs 10 times in the dataset. For this graph, we divide the dataset in 10 random sample chunks, each containing 10% of the dataset. Each of the separate chunks contains around 65K unique keys. This does not give much information about the NDV of the full dataset (i.e., 100K). However, more information becomes visible when comparing sample chunks, like investigating how many new (i.e., never-before-seen) keys are observed in an additional chunk given the context of the sample chunks seen before. In Figure 1, this means 65K new unique keys for the first sample, 24K new unique keys, when adding the second sample, 8K new unique keys for the third sample, and so on. This reduction rate is a good indicator on the key distribution of the entire dataset, so we want to use it for our feature creation. With the chunking approach, it is easier for the ML model to extract information by comparing n chunks to $n + 1$ chunks, compared to a single sample approach with the same size of $n + 1$ chunks.

Generally, our approach can be used with a variable number of chunks and chunk sizes. However, for this paper, we define the chunk size to 0.5% of the dataset, while using three chunks in total, leading to a full sample size of 1.5% of the dataset.

4.1.2 ***Input Features.***
To construct the model features, we are using the multiple sample-chunks idea as introduced above. Features are either based on single chunks or a combination of chunks, e.g., groups of two or three chunks treated as a combined sample.

For some of the used features, we construct a frequency histogram of the data distribution for a chunk or groups of chunks. The histogram is similar to the approach in Section 3.3 (without normalization). For each key, the frequency of its occurrence in the sample is calculated, before aggregating the frequencies to the frequency histogram. For example, if a sample contains 100 keys and the frequency histogram of this sample only contains $f_1 = 100$, we know that each key occurs only once, i.e., each key is unique. On the other side, if $f_{100} = 1$, we know that there is only one unique key occurring 100 times in the sample (frequency of 100).

To create features out of the sample chunks and frequency histograms, we compose a large amount of features and feature combinations and apply feature selection [19] to reduce the set of features to important ones for the prediction. The resulting feature set contains 14 features, which can be grouped into three categories.
Category 1: Features based on the frequency histograms:

(1.1) Amount of keys with frequency 1 based on a single chunk
(1.2) Amount of keys with frequency 2 based on a 2 chunk group
(1.3) Amount of keys with frequency 1 based on 3 chunk group
(1.4) Amount of keys with frequency 2 based on 3 chunk group
(1.5) Amount of keys with frequency 3 based on 3 chunk group
(1.6) Amount of keys with frequency 4 based on 3 chunk group

Six features are based directly on the frequency histograms. Most features are using the three chunk group, because this group contains the largest sample (three times 0.5%), hence it uses the most amount of data containing the most information compared to single chunks or two-chunk groups.

Category 2: Features based on unique keys and chunk differences:

(2.1) *Amount of new unique keys in the second chunk when considering the first chunk*
(2.2) *Amount of new unique keys in the third chunk when considering the first two chunks*
(2.3) *Total amount of unique values (NDV) for 3 chunk group*

There are two features looking at the additional unique keys that can be observed by adding one chunk to one or two previous chunks and one feature that is using the NDV of the three chunk group.

Category 3: Feature Combinations:

(3.1) *Feature 2.2 divided by Feature 2.1*
(3.2) *Feature 2.1 divided by Feature 1.4*
(3.3) *Feature 2.2 divided by Feature 1.4*
(3.4) *Feature 1.2 divided by Feature 1.5*
(3.5) *Feature 2.3 divided by population size.*

There are five features either using a division of the previous features or a division by the population size. These features are determined as important by feature selection [19] and extensive testing, given hundreds of similar feature combinations as initial input.

4.1.3 *Averaging Chunk-based Statistics*.

In the previous section, we introduced features either based on single chunks or groups of chunks, e.g., groups of two or three chunks treated as a combined sample. To add robustness to our features, we average the chunk and chunk group statistics, whenever there are multiple possible combinations.

For the single-chunk features, there are exactly three chunks to choose from so we compute the feature for each chunk and average the results. For the two-chunk-group features, there are three different combinations of two chunks, so we compute the feature for each combination and average the results. For the three-chunk-group there is only one possible combination. For all cases, where we have multiple possible combinations of chunks, we look at every combination, extract our features, and average the feature values over similar combinations. This averaging approach helps to mitigate the randomness of the sampling.

4.1.4 *Feature and Label Normalization*.

So far, the features taken from the sample chunks are absolute values or averages of absolute values. To make our approach generalized for dataset sizes that have not been seen by the model, we need to normalize them depending on the sample size. Therefore, we divide the features by the total number of keys in the chunk or chunk-group, on which the feature is based on. For example, if a chunk contains 200 keys and there are 100 keys that occur only once, then Feature 1.1 is using the normalized value of 0.5 instead of the absolute value of 100. This allows our approach to identify similar pattern for datasets with completely different numbers of keys, since after normalization, features are in the same range of values. This kind of normalization is applied to all features of Category 1 and 2, but not Category 3, since there the features are already normalized through the inherent division.

A similar normalization is done for the prediction label (i.e., the NDV). There, instead of predicting the NDV directly, we follow our idea of feature-normalization and predict the *relative NDV* ($\frac{NDV}{population\_size}$) instead. This has two advantages, as datasets with different NDV and different number of elements might be similar through their relative NDV, and the ML target values are limited to numbers between 0 and 1. The latter is a large advantage as most ML algorithms can only predict labels that lie within the range of labels seen in the training data.

Using labels and features in a normalized format allows us to predict for arbitrary datasets, even if they are not within the boundaries of our training data. During inference, the model predicts relative NDVs, so for a final result the prediction needs to be multiplied by the population size.

4.2 Model Optimizations

In addition to our general ML approach, we present two optimizations to improve the predictions for low and high relative NDVs.

4.2.1 *Label Transformation*.

As stated before, we are normalizing prediction labels (NDVs) with the population size to predict relative NDVs instead of absolute NDVs. There could be large relative NDVs close to and including the value 1.0, which represent an NDV close to the population size of a dataset, hence, most of the values are unique. There could also be low relative NDVs close to (but not including) the value 0.0, which represent a NDV with a small number of unique values.

Many ML algorithms internally use a scoring metric like mean-absolute-error (MAE) or mean-squared-error (MSE), which are optimized to penalize large errors, while being willing to allow smaller ones. With our approach of using a relative NDV as target, this might cause problems for small relative NDVs. As an example, for a Dataset A with 100M keys and a NDV of 1M, the target value is 0.01. For Dataset B with 2M keys and an NDV of 1M, the target value is 0.5. If the model is using a metric like MSE, it might consider an error of $+0.1$ as acceptable. For Dataset A, this implies a percentage error of 1000%, while for Dataset B, with exactly the same NDV, this only causes an error of 20%. In general, we found that a percentage error or mean-absolute-percentage-error (MAPE) is more suited as an optimization goal for our purposes. However, many ML algorithms inherently do not support this optimization metric, so we have to use MAE or MSE.

To avoid this imbalance and allow smaller relative NDV to be predicted in a good quality, we transform our label using logarithmic transformation. Before training, we apply a logarithmic operation to all labels (y) and then train the algorithm with the new label ($y_{log} = log(y)$). During inference, the model predicts the log-scaled label, so the actual prediction needs to be transformed again using the constant e to the power of the log-scaled prediction ($y = e^{y_{log}}$). With this transformation, very small labels are transformed to larger negative values and prediction errors on these values have a larger magnitude than before, hence, are more prioritized in the optimization of the ML algorithm.

4.2.2 *Edge-Case Model*.

In addition to the label transformation, we noticed that NDV edge cases need more optimization as they are hard to predict exactly

F_1	F_2	F_3	F_4	label	class
1	2	10	20	10	A
2	5	20	30	20	A
11	2	10	40	40	B
12	3	20	50	50	B
1	7	20	40	30	X

Table 1: Features values can be used as label. Here two patterns emerge (A) when $F_1 \leq 2$ and $F_2 \leq 5$ then $label = F_3$ and (B) when $F_1 \geq 11$ then $label = F_4$. (X) symbolizes no match.

right for the presented regression model. Such edge cases are either very small NDVs, where the NDV observed in the sample is close to the NDV of the full dataset; or large NDVs close to or equal to the population size. We further noticed for these cases, that the actual prediction target is represented in one or multiple of the input features. For example, for very small NDVs, where all unique keys of the dataset can be seen in the sample, Feature 3.5 equals the relative target NDV. Additionally, when all keys of the dataset are unique, Feature 1.1 and 1.3 have the value 1.0, which also equals the relative target NDV in this case.

To detect these cases automatically, we construct a ML model to predict when a Feature F_i equals the label and thus can be used directly as result. An example for this problem is shown in Table 1. There Feature F_3 equals the label but only if Feature F_1 is less or equal to 2 *and* Feature F_2 is less or equal to 5.

The presented ML regression model is not suited for finding these cases and in general it is not common for ML algorithms to conditionally use an exact feature value as prediction result. To solve this problem, we create a classification ML model, where we encode certain patterns in the data (like (A) and (B) in Table 1) as separate classes. In detail, we apply the following steps:

(1) With the given training data, we check if a feature is equal to the prediction label (relative NDV). For training data, both, the features and the label are known.

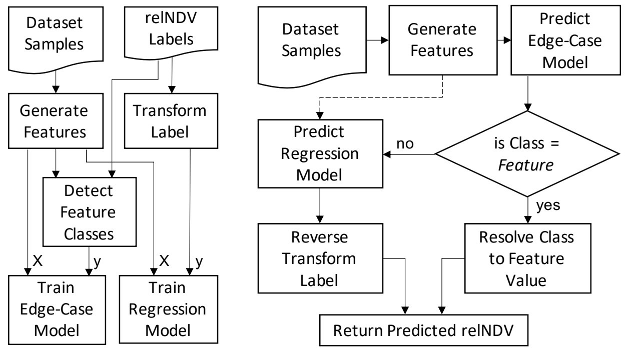
(2) We assign classes for each instance, where features equal the label. In our example from Table 1, this results in Class A ($label = F_3$) and Class B ($label = F_4$). The classes only describe the observable outcomes ($label = F_i$), but do not know the reason or pattern behind it. The number of classes depends on the number of features that, for some datasets, equal the label, with an additional Class X for the remaining cases (no matching feature). For the example in Table 1 this results in three different classes.

(3) Based on the created classes, the classification model uses the same features as the regression model, however, using the classes as prediction label instead of the relative NDV values. We only provide the classes to the model. The model itself finds the underlying pattern, when these classes (i.e. label matches) occur.

(4) During the inference, the model predicts a class for every instance. For each predicted class, we either convert the class to a result value by replacing the class with the corresponding feature value or use the described regression model to predict the NDV if the predicted class indicated that no feature is matching.

4.3 Model Training and Inference

Given our general ML approach and the proposed optimizations, Figure 2 is illustrating the model training and inference steps.



(a) ML Training **(b) ML Inference**

Figure 2: Training and inference for the ML approach.

For **model training**, we use dataset samples with the corresponding relative NDV numbers of the full dataset (relNDV label). The samples are used for generating the features as described in Section 4.1. The regression model is trained with the generated features and the log-transformed NDV labels. The Edge-Case Model first needs to detect and encode features classes based on the generated features and the relative NDV labels. The Model then is trained using the detected classes and the generated features. We evaluated multiple ML algorithms for our two models and found that ensemble models based on multiple decision-trees are suited best for this task. As a result, we are using a Random Forest [2] algorithm for our regression model and a AdaBoost [13] algorithm for the Edge-Case Model. The ML model training is performed off-line, which means it is trained outside production environment, where we have time and resources for extended model optimization. Only the trained and optimized models are deployed.

For **model inference**, we generate features in the same way as in the training step. The features are used to first predict a feature class with the Edge-Case model. If the predicted class is a feature, we resolve this class by using the corresponding feature value and return the relative NDV. If the predicted class is indicating no feature match, then the generated features are used with the regression model and the reverse log-transformation to predict the final relative NDV.

Depending on the prediction problem, the real NDV numbers need to be transformed into relative NDV values ($relNDV = \frac{NDV}{population\_size}$) for the training step and the predicted relative NDVs need to be transformed back into real NDV values ($NDV = relNDV \cdot population\_size$).

5 EVALUATION

In this section, we evaluate our two approaches together with the state-of-the-art described in related work. First, we describe the evaluation setup used in the experiments and then we present a general comparison of the different approaches, followed by a more detailed investigation for our approaches.

5.1 Evaluation Setup

Our evaluation setup consists of the evaluation datasets and their generation, algorithms, and error metrics.

5.1.1 *Datasets*.

To compare the different approaches, we generate a number of different dataset corpora. We only use datasets with a population size of 100K keys or more, because smaller datasets can be completely scanned to compute the NDV without high overhead. In addition to datasets with specific distributions, we create larger corpora of datasets from common database benchmarks and open data sources, as these are relevant examples for database systems. Finally, we generate a large dataset corpus based on random data to train our ML approach. The ML model is only trained on this corpus, which does not include any other datasets we test on. The details of the datasets are the following:

- The ***Uniform*** corpus contains 7 datasets, each with a population size of 10M, where values occur uniformly 1, 2, 3, 4, 5, 10, 100, and 1000 times, resulting in relative NDVs from 0.1% to 100%.
- The ***Zipf*** corpus includes 11 datasets generated from a standard Zipf distribution [38] with a infinite vocabulary, so that the probability of sampling key i is given by $f(k) = \frac{k^{-s}}{\zeta(s)}$, where $\zeta(s)$ is the Reimann Zeta function, and s is a parameter governing the skewness of the data. The population size was set to 10M and s was set to values $s \in \{1.01, 1.1, 1.2, \ldots, 2\}$. The observed relative NDVs range from 0.04% to 70%.
- The ***dZipf*** (discrete Zipf) corpus includes 20 datasets having the following Zipfian property: if the keys are arranged in decreasing order of frequency, the frequency of the k^{th} order key is proportional to k^{-s}. Each of the 20 datasets was generated by a unique value from the set $s \in \{0.1, 0.2 \ldots 2.0\}$. The target population size was set to $N = 10M$, and the number of distinct values D was set so that the lowest order key had a frequency of 1. The value of D meeting this condition was found by numerically finding the largest D such that $H_{D,s} \cdot D^s \leq N$, where $H_{D,s}$ is the Harmonic number $H_{D,s} = \sum_{i=1}^{D} \frac{1}{i^s}$.
- The ***TPCH*** corpus is taken from TPCH database benchmark tables [35], using the scale factor 1024. The benchmark tables contain a total of 61 columns, however, only 54 columns have more than 100K rows. The largest columns contain 6.1B rows.
- The ***TPCDS*** corpus is taken from TPCDS database benchmark tables [20], also using a scale factor of 1024. There, 206 columns contain more than 100K rows (out of 429 columns in total). The largest columns contain up to 2.9B rows.
- The ***RWD*** (RealWorldData) corpus is constructed using 10 real-world data sources from the cities of Seattle and New York [21–30]. The sources consist of multiple tables and result in 340 columns with more than 100K rows. The largest columns contain about 62M rows.
- The ***MLtrain*** corpus was specifically created to train the ML models for the ML approach. It contains 100K datasets with a population size between 100K to 10M. Algorithm 1 shows the algorithm used to generate the MLtrain datasets. The goal of the algorithm is not to randomly generate a full dataset but to generate a random frequency histogram that represents a dataset. First a target population size is defined as a random number between 150K and 10M (Line 2). This target population size is reduced with every iteration until it is smaller than 50K. With every iteration (Line 3), an initial frequency is set to the population size (Line 4). Afterwards, this frequency is reduced (Line 7) by multiplying it with a random floating point number between 0 and 1. This reduction is performed between 1 and 9 times (Line 5). After defining the frequency, the amount (how often this frequency occurs) is chosen randomly between 1 and the remaining population size divided by the chosen frequency (Line 9). Finally, the population size is reduced by the chosen frequency multiplied with the chosen amount (Line 10) and both values are updated in the frequency histogram (Line 11). The iterations are repeated until the population size is below 50K. To produce the MLtrain dataset corpus, this algorithm is executed 100K times. It is important to check if exactly matching datasets have been created, as these need to be deleted in order to allow correct leave-out cross-validation.

Algorithm 1 Dataset generation for MLtrain dataset corpus

```
 1: procedure GENERATE_DATASET
 2:   $popSize \leftarrow genRandomInt[150K, 10M]$
 3:   while $popSize > 50K$ do
 4:     $frequency \leftarrow popSize$
 5:     $n\_reductions \leftarrow genRandomInt[1, 9]$
 6:     while $n\_reductions \neq 0$ do
 7:       $frequency \leftarrow frequency * genRandomFloat(0.0, 1.0]$
 8:       $n\_reductions \leftarrow n\_reductions - 1$
 9:     $amount \leftarrow genRandomInt[1, popSize/frequency]$
10:     $popSize \leftarrow popSize - (frequency * amount)$
11:     $update\_frequency\_histogram(frequency, amount)$
```

Table 2 summarizes the properties of the dataset corpora. To categorize the datasets, we investigate their frequency histograms. We use the standard deviation of observed frequencies divided by the mean frequency as a measure of data uniformity. Uniform datasets have a value close to 0 as only one frequency is observed (e.g., frequency 10 for cases where each key occurs 10 times). Less uniform datasets (like Zipf distribution) have a higher value. As we can see from Table 2, TPCH has more uniform datasets, while TPCDS and MLtrain have more datasets that are less uniform. The relative NDV statistics show that TPCH has both low and high NDVs, while TPCDS, RWD, and MLtrain have mainly small NDVs.

For sampling, we always take a 1.5% random sample, if not otherwise stated. The percentage is based on the default input size of our ML approach (Section 4.1.1) and we discuss this choice of percentage further in Section 6. We differentiate the sample type between sampling with replacement (*i.i.d.* or *independent and identically distributed*) and sampling without replacement (in this paper marked as *non i.i.d.*). For our tests, exactly the same sample is given to all approaches for the prediction.

		relative SD		relative NDV		
	#datasets	#<0.5	#≥0.5	#<1%	#>95%	#rest
Uniform	7	7	0	1	1	5
Zipf	11	0	11	6	0	5
dZipf	20	1	19	8	1	11
TPCH	54	41	13	30	12	12
TPCDS	206	54	152	175	6	25
RWD	340	31	309	296	9	35
MLtrain	100000	4036	95964	70859	133	29008

Table 2: Properties of the datasets used in experiments. The relative standard deviation (SD) is the standard deviation of observed frequencies divided by the mean frequency.

	Corpus (n_datasets)	Uniform (7)		Zipf (11)		dZipf (20)		TPCH (54)		TPCDS (206)		RWD (340)		MLtrain (100K)	
	Sampling Type	*i.i.d.*	*n.i.i.d.*	*i.i.d.*	*n.i.i.d.*	*i.i.d.*	*n.i.i.d.*	*i.i.d.*	*n.i.i.d.*	*i.i.d.*	*n.i.i.d.*	*i.i.d.*	*n.i.i.d.*	*i.i.d.*	*n.i.i.d.*
error ratio	Hybrid	4.62	4.68	1.05	1.06	1.76	1.79	2.53	2.55	2.03	2.04	1.92	1.92	3.94	3.98
	Charikar AE	1.05	1.28	4.52	4.27	2.59	2.45	1.08	1.09	1.17	1.17	2.11	2.09	1.82	2.05
	HNE	1.07	1.29	2.02	1.94	1.61	1.46	1.04	1.04	1.13	1.13	1.56	1.55	1.50	1.49
	ML approach	1.02	1.26	1.76	1.59	1.39	1.35	1.03	1.04	1.06	1.16	1.56	1.54	1.25	1.24
MAPE	Hybrid	361.6%	368.0%	5.5%	5.8%	76.2%	79.0%	152.7%	155.1%	102.6%	104.0%	79.3%	80.2%	277.3%	281.6%
	Charikar AE	5.1%	28.2%	76.5%	74.1%	55.6%	52.7%	4.0%	4.5%	10.1%	10.7%	73.3%	75.1%	41.1%	68.7%
	HNE	6.5%	28.5%	61.8%	67.6%	37.8%	31.6%	3.1%	3.6%	10.1%	10.7%	42.9%	42.2%	23.8%	26.3%
	ML approach	1.8%	25.7%	39.1%	32.3%	24.6%	24.5%	2.9%	4.0%	5.0%	6.6%	25.3%	24.4%	17.2%	16.7%

Table 3: Overall results for different approaches and different dataset corpora, for *i.i.d.* and *non i.i.d.* (*n.i.i.d.*) sampling. ML model is being trained on the corresponding MLtrain data (*i.i.d.* or *non i.i.d.*). In the upper half the error is calculated as error ratio (eq. (22)) and in the lower half the same error is shown as MAPE.

5.1.2 *Algorithms*.

For the quality comparison, we use the two approaches presented in this paper and compare them to two approaches from related work. The evaluated approaches are, in detail:

- The ***Hybrid*** approach based on a combination of Shlosser [33] and Smoothed Jackknife [15]. Combining both approaches was proposed by Haas et al. [15].
- The Adaptive Estimation approach (***Charikar AE***) proposed by Charikar et al. [5].
- Our Histogram Normalization Estimator (***HNE***) presented in Section 3.
- Our ***ML approach*** based on two machine learning models as presented in Section 4.

5.1.3 *Metrics*.

As comparison metrics, we use error ratio (eq. (22)) and Mean Absolute Percentage Error (MAPE). Both metrics characterize the relative error of the different approaches and a lower value symbolizes a smaller error.

Additionally, we introduce a new metric to differentiate between over-prediction and under-prediction, because the differentiation is not possible in Error Ratio or MAPE. We call this metric *signed relative error (sRE)*, which is similar to error ratio (eq. (22)) with changes to make the result signed and based on 0 instead of 1:

$$sRE = \begin{cases} 1 * \left(\frac{\text{Estimate}}{\text{Target}} - 1\right), & \text{if Estimate} > \text{Target} \\ \\ -1 * \left(\frac{\text{Target}}{\text{Estimate}} - 1\right), & \text{if Estimate} \leq \text{Target} \end{cases} \tag{23}$$

With this metric, over-prediction has a relative error above zero, under-prediction a relative error below zero, and zero itself symbolizes no error. Please note, that *sRE* can only be aggregated for positive and negative results separately, as differently signed errors might cancel each other out.

5.2 General Evaluation

Table 3 shows the error ratios and MAPE scores for the different approaches using the datasets presented in Section 5.1.

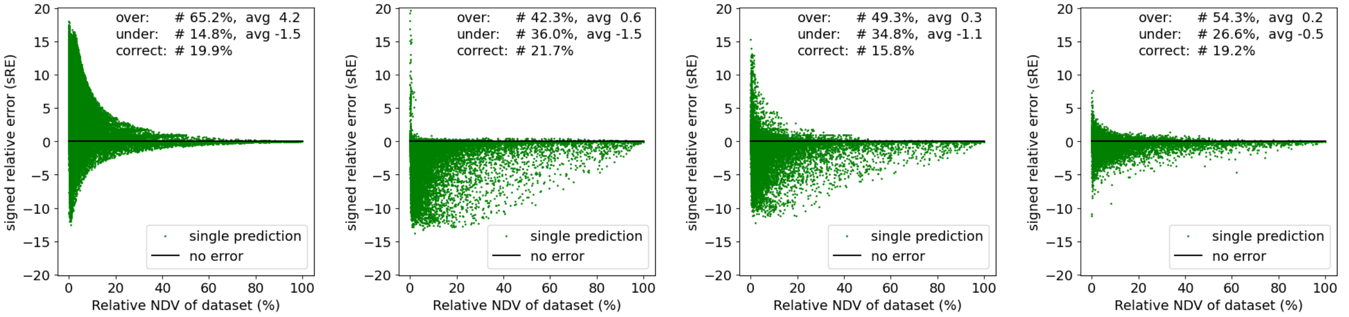
The Hybrid approach performs well for the Zipf datasets in experiments, while producing larger errors for the other datasets. This illustrates the statement made by Haas et al. [15] that it is nearly impossible to have an approach work well with all distributions. Here, the internal Shlosser algorithm [33] is highly tuned to skewed datasets like Zipf and dZipf. The type of sampling does not seem to impact results for this approach.

The Charikar AE approach shows good results for uniform data and the benchmark datasets, but shows worse results for Zipf distributed data, RWD, and the MLtrain dataset corpus. In general, it outperforms the Hybrid approach except for the Zipf corpus. For the Zipf-like datasets, it performs better on MAPE, compared to the error ratio (e.g., 50% MAPE ideally corresponds to an error ratio of 1.5). This is caused by strong under-prediction, which causes the MAPE to show a 100% error in the worst case, while the error ratio can show a much higher value. *i.i.d.* sampling is better for Charikar AE for Uniform and MLtrain datasets with MAPE differences of up to 5.5x compared to *non i.i.d.* sampling.

Our HNE based approach is either similar or significantly better than the Charikar AE approach, which becomes especially visible for the error ratios of the Zipf-like datasets and RWD. The difference between MAPE and error ratio is not as strong as the Charikar AE approach, which illustrates that the under-prediction problem is less pronounced. Except for the uniform corpus, we do not see a significant preference for *i.i.d.* or *non i.i.d.* sampling.

Finally, our ML approach shows good results for all corpora, except for the the Zipf datasets, where it outperforms all approaches except the Hybrid approach. For all the experiments, the model is trained on the full MLtrain dataset. When predicting the MLtrain dataset itself, the model is trained using 10-fold leave-out cross-validation (CV). This means it is trained on 90% of the datasets, while only predicting for the remaining 10%. This is shifted 10 times until the NDV for all datasets is predicted. Using CV avoids that information about the test data is used for training the model. For the RWD corpus, the ML approach outperforms the other approaches, with a 3x MAPE reduction compared to Charikar AE. Interestingly, the ML approach has a similar error ratio as HNE but a much better MAPE, caused by slight over-prediction for HNE and slight under-prediction for the ML approach. The error is generally a bit higher for RWD compared to most other copora, due to containing many zipf-like datasets as indicated in Table 2. The ML approach is flexible in using the *i.i.d.* and *non i.i.d.* data; however, for the uniform corpus it shows a similar behavior as Charikar AE and HNE, where *i.i.d.* is much better than *non i.i.d.* .

To summarize, both of our approaches improve upon the current state-of-the-art in general, while the ML approach shows the best results over all. The only exception is the Hybrid approach, which is specifically optimized for highly skewed datasets (Zipf), while showing larger errors for all other copora.



(a) Hybrid approach **(b) Charikar AE approach** **(c) HNE approach** **(d) ML approach**

Figure 3: Prediction errors (sRE, eq. (23)) for the MLtrain dataset using *i.i.d.* samples ordered by the relative NDV of the datasets.

5.2.1 *Detailed Evaluation on MLtrain Datasets.*

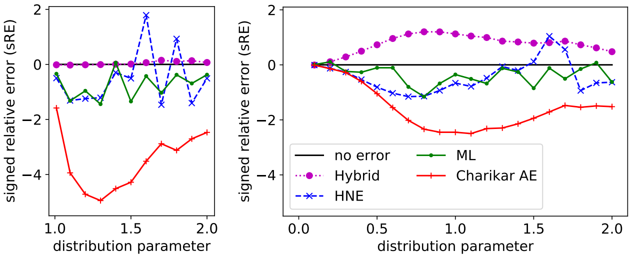
As the MLtrain corpus provides us with a large number of datasets, we investigate the predictions for these datasets in more detail. Figure 3 shows the predictions for all four approaches as a scatter plot using our *sRE* metric (eq. (23)). The true relative NDV of the dataset is on the x-axis and the *signed relative error* on the y-axis. Predictions closer or equal to zero sRE are better. In each figure, there are 100K dataset predictions with the majority of predictions being for low relative NDVs (Table 2), i.e., on the left side of the graph. Additionally, we added statistics about the percentage of the predictions, which are over-predicted (*over*), under-predicted (*under*), or correctly predicted (*correct*), together with error aggregations.

The first observation is the shape of the scatter plot. The Hybrid approach has the most errors for low relative NDVs with a strong tendency to over-predict. The average sRE for over-predicted datasets is 4.2 (equal to an error ratio of 5.2). The Charikar AE approach has also the majority of errors as over-prediction, however, these predictions are much closer to the correct NDV, resulting in an average sRE of 0.6. The under-predictions have the same sRE as the Hybrid approach, while occurring for small *and* large relative NDVs. Our HNE approach reduces the average error for over-prediction and under-prediction compared to both previous approaches, while the general shape of the plot looks similar to Charikar AE. The ML approach shows a much tighter scatter plot with a tendency to over-predict with a sRE of 0.2, while also only having an under-prediction sRE of only -0.5.

When looking at the exactly matching estimations, the Hybrid approach (19.9%) and the Charikar AE approach (21.7%) have the most amount of correct predictions compared to the HNE approach (15.8%) and the ML approach (19.2%). This means that for HNE and the ML approach, many predictions are close to the correct values and few predictions are exactly correct. This influences the average sRE numbers to be closer to 0 than for the other approaches, as more predictions are included into the average. However, the shape of the scatter plot and the results in Table 3 show that these approaches are performing well and that this is not solely the effect of these non-correct predictions.

5.2.2 *Detailed Evaluation on Zipf Datasets.*

In general, our approaches show good results for all the datasets in Table 3. But specifically for the Zipf dataset corpus the Hybrid approach performs much better. Therefore, we illustrate the results of the Zipf and dZipf corpus in detail in Figure 4. The datasets are plotted according to the distribution parameter used during their creation, while showing the signed relative error (sRE).

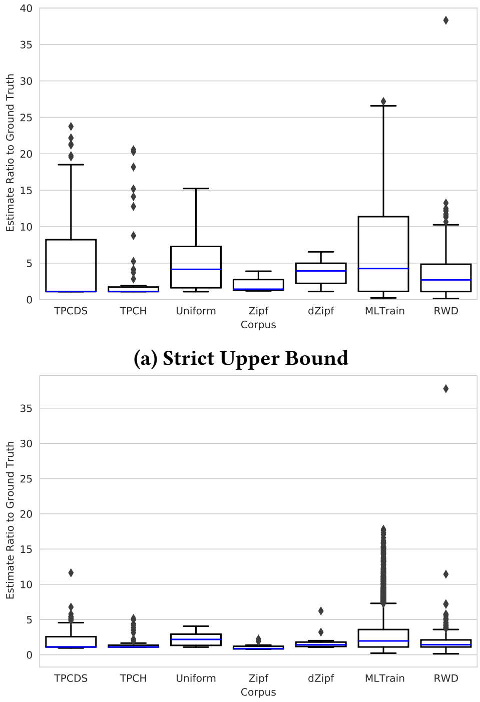


(a) Zipf corpus **(b) dZipf corpus**

Figure 4: Zipf-like distribution using *i.i.d.* sampling.

Figure 4a shows that the Hybrid approach is indeed nearly always close to zero, indicating near-perfect predictions. The Hybrid approach consists of two internal algorithms, one of which is chosen automatically based on the data distribution in the sample. For the Zipf datasets, the chosen algorithm internally is always the Shlosser approach [33], which seems highly optimized for exactly this Zipf distributions. The HNE and ML approaches are close to the Hybrid approach, however, often predicting around half of the actual NDV (*sRE* of -1). The HNE approach also seems to alternate between over-prediction and under-prediction depending on the Zipf distribution parameter. The Charikar AE approach constantly under-predicts by a large margin, with an sRE of -5 in the extreme case.

For the HNE and ML approach, the behavior is similar in Figure 4b for slightly less skewed Zipf-like data. Charikar AE shows a better sRE, while still under-predicting more than the other approaches. The Hybrid approach, however, shows a much higher error resulting in often over-predicting about 1x. This shows that the internal Shlosser model [33] is highly optimized for the specific distribution shown in Figure 4a but performs worse for any deviation from this target distribution.



(a) Strict Upper Bound

(b) Upper Bound based on Geometric Mean

Figure 5: Distribution of the bound width (using bounds provided by HNE) on dataset corpora, for 1.5% *i.i.d.* samples. The value is divided by the NDV ground truth for normalization.

5.3 Upper Bound on NDV Estimates

As previous theoretical negative results [5, 6] prove, error ratio bounds on NDV estimates are expected to be large, as any sample data could have been drawn one of two extreme distributions: either the sample NDV could be the dataset NDV, or the dataset NDV could be an extremely high value due to presence of singletons. As a result, while two-sided bounds on NDV estimates can be provided (by using the sample NDV as the lower bound, and the singleton estimate as the upper bound), they are often too broad to be useful.

With HNE we mainly address the upper-bound problem. There, the goal is to be as close to the ground truth as possible, while avoiding under-estimation. We do not provide a lower bound, but use the natural lower bound being the sample NDV. The upper-bound estimation has many applications in database systems. For example, a query optimizer might have a threshold NDV value, where a certain plan is only chosen above a certain value. Additionally, NDV estimates may be used for memory allocation and risks from under-allocation might mean that the system would prefer to over-allocate memory based on an upper bound.

As described in Section 3.3.2, such an upper bound is provided by eq. (21). Figure 5a shows the distribution of the ratio of the upper bound to the NDV ground truth across test corpora. The upper bound correctly over-estimates the NDV ground truth in almost all cases (as the upper-bound to ground truth ratio is above 1). There are only 15 datasets, all in the MLTrain corpus (0.015%), which do not over-estimate the NDV.

Optimizations		# affected	MAPE		gain
(from	→ to)	datasets	before	after	
RM	→ RM+ECM	7151	2.66%	0.34%	7.8x
RM	→ RM+yT	78858	28.22%	21.22%	1.3x
RM+yT	→ RM+yT+ECM	7044	1.38%	0.34%	4x

Table 4: Improvements using ML optimizations tested with Regression Model (RM), Edge-Case Model (ECM), and label transformation (yT).

An interesting aspect of Figure 5a is the set of outlier values that represent a high over-estimate, particularly for TPCH. This is in line with the scenario mentioned above, where the sample NDV is actually the dataset NDV. Technically in these cases, the guarantee of an over-estimate still holds.

Overall, the strict upper bound is generally high, compared to the ground truth. For this reason, as discussed in Section 3.3.2, we propose the use of the geometric mean of the HNE and upper-bound NDV estimates, as an alternative upper bound. Figure 5b shows the ratio of the geometric mean based upper bound estimate to the ground truth. While the geometric mean (GM) upper bound is less strict, in practice, it over-estimates the ground truth NDV in almost all cases. The GM upper bound to ground truth ratio is greater than 1 for all histograms across all datasets, with the exception of the Zipf, MLTrain, and RWD datasets. For the Zipf dataset, the under-estimate occurs for about 60% of datasets. However, the under-estimate is small in magnitude: the ratio of estimate to ground truth is 0.8 in the worst case. Besides this, the GM bound under-estimates in 3% of histograms in the MLTrain dataset, and 4% of histograms in RWD. Note, however, that the GM bound has a much narrower range. For a large number of histograms across all corpora, the bound is within 2.5x of the ground truth. In situations where the system is robust to occasional under-estimates, the GM bound can provide a reasonably narrow upper bound estimate.

5.4 Optimization Impact for ML approach

In this part, we illustrate the impact of our optimizations for the ML model (Section 4.2). We evaluate the general Regression Model (RM, Section 4.1) separately and add two optimizations, (1) label transformation (yT) and (2) the Edge-Case Model (ECM), one at a time (Section 4.2). Table 4 shows the results of these experiments with the MLtrain dataset corpus using cross validation. We can see, that the Edge-Model affects about 7% of the datasets, which is expected since this model is designed to handle the edge cases and not the majority of the datasets. This means that for 93% of the datasets, this model will predict the 'no-match' class indicating that the regression model should be used. When only looking at the 7% affected datasets separately, the MAPE changed from 2.66% to 0.34%, showing that the regression model handled these cases already reasonably well, but the Edge-Case Model still reduces the errors of these cases by about 8x. The impact of the label transformation (yT) is only 1.3x error improvement, however, this optimization is affecting the majority of the datasets (79%). Interestingly, the label transformation and the Edge-Case Model improve similar kinds of datasets. This can be seen in Table 4, where adding the Edge-Case Model to the Regression Model with label transformation shows only an improvement of 4x (instead of 7.8x) for the affected datasets.

6 DISCUSSION

In this section, we want to discuss different perspectives of judging our approaches for the final question: which approach is the best?

6.1 Estimation Quality

As shown in Section 5.2, both, the ML approach and HNE approach show a robust performance without larger errors as seen with the other approaches. In general, the ML approach surpasses the HNE approach, but the errors of both are comparably low.

6.2 Varying the Sampling Percentage

One of the main limitations of the ML model is the fixed sample percentage. The ML model is specifically trained on the 1.5% sample size. While it is expected that the model will do reasonably well with smaller or larger sample percentages, it needs to be trained for every new scenario. This means that the ML approach either supports only one fixed sample percentage or that several models need to be trained and deployed for a number of fixed percentages.

In contrast, HNE supports a variable sample percentage, since the percentage is an input to the model. The percentage can be adjusted for each separate prediction, allowing further optimizations. One optimization could be upper sampling limits, where the algorithm never samples more than an absolute number of keys. For example, while 1.5% could be the default sampling percentage, the system might have an absolute upper limit and not sample more than a certain number of keys for performance reasons. This is especially useful for runtime critical applications. Another use case is adaptive sampling, where the model starts sampling with a small percentage and increases the percentage if the bounds indicate high prediction uncertainty. Generally, having upper and lower bounds is also a feature of HNE, which is not provided by the ML approach.

6.3 Maintainability

We have seen in Section 5.2 that models perform differently on different datasets and further adjustments might be needed for different applications. For example, both of our approaches need to be improved if the target datasets are mainly consisting of Zipf distributed keys.

For the HNE approach, this involves changing and extending the theoretical principles of the model itself and certainly needs an expert in the field to do so. On the other side, the ML approach can be trained with data that is targeted by the application. So it is possible to specifically generate or observe target data to train the ML model. The model could even be trained *online*, where it predicts the NDV for certain datasets, while at a later stage it gets the real NDV as feedback. This can be used to automatically specialize the model without changing any core principles.

6.4 Applicability

Finally, the question arises on how easy can the approaches be deployed and applied to existing applications.

For the ML approach, specific libraries are needed, which have to be present in the product. This involves licensing of these libraries and it prevents the model from being deployed in specific sand-boxed environments like SQL-based stored procedures in a database system. Additionally, the trained model itself has a significant memory footprint. Our ML model contains hundreds of underlying decision tree structures and when stored to disk, it results in about 200MB of compressed model-internal data. This might make it unusable for environments with limited resources.

On the other side, the HNE approach does not need specific libraries and mainly consists of a few hundred lines of code. This is much easier to deploy in limited environments and can be ported easily to any target programming language.

6.5 Which Approach is the Best? ... It depends!

Judging from the prediction quality and maintainability, the ML approach should be preferred. However, this is only possible if the dependency on libraries and fixed sample percentages does not pose a limitation for the final application. Seeing that the HNE approach has a similar prediction quality, is easier to integrate to existing projects, and supports variable sample percentages, it seems that this approach is more flexible in its application. Especially the latter point is important since sampling percentages are usually preferred smaller than 1.5% with the option to sample more if the prediction is not good enough. In the end, it mainly depends on the application environment and the usage of the NDV predictor.

7 CONCLUSION

In this paper, we investigated the problem of distinct value estimation based on a dataset sample. We proposed two novel approaches using different methods, a statistical method and a Machine Learning based method. Both our approaches outperform the competitors, though performing worse for very specific datasets, which other approaches are specifically optimized for. Overall, the ML based technique performs best, with up to 3x in average error reduction for real-world datasets. However, better prediction quality does not mean that this approach can be applied directly to existing projects. It rather depends on the project specifics and it might be better to choose a statistical method for easier integration. This is the main reason why we explore two inherently different approaches in this paper.

In future work, we plan to extend the HNE approach into an adaptive sampling framework, where the algorithm starts with a small sample size, and the sampling percentage is increased until a certain exit criteria is reached. This can result in faster processing because large samples are only taken where required. We also plan to investigate extending HNE to include information from multiple samples, like the ML approach. A key problem statistical estimators face is that the number of singletons in the dataset are not known. Incorporating the rate at which new keys are observed with each additional sample could address this problem.

The ML approach can always be extended by more targeted training data for certain distributions (e.g., Zipf) or additional features. Additionally, the whole approach could be extended by incorporating other approaches like HNE or the Shlosser estimator [33] in the model. Currently the Edge-Case model is deciding to use a certain feature value or the regression model. This makes it easily extensible to add more models and let the edge-case model decide, which one to use for a given data sample.

REFERENCES

[1] R. Barber, G. Lohman, I. Pandis, V. Raman, R. Sidle, G. Attaluri, N. Chainani, S. Lightstone, and D. Sharpe. 2014. Memory-Efficient Hash Joins. *Proc. VLDB Endow.* 8, 4 (Dec. 2014), 353–364. https://doi.org/10.14778/2735496.2735499

[2] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[3] J. Bunge and M. Fitzpatrick. 1993. Estimating the Number of Species: A Review. *J. Amer. Statist. Assoc.* 88, 421 (1993), 364–373. http://www.jstor.org/stable/2290733

[4] K. P. Burnham and W. S. Overton. 1978. Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals. *Biometrika* 65, 3 (1978), 625–633. http://www.jstor.org/stable/2335915

[5] 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* (Dallas, Texas, USA) *(PODS '00)*. Association for Computing Machinery, New York, NY, USA, 268–279. https://doi.org/10.1145/335168.335230

[6] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1998. Random Sampling for Histogram Construction: How Much is Enough?. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data* (Seattle, Washington, USA) *(SIGMOD '98)*. Association for Computing Machinery, New York, NY, USA, 436–447. https://doi.org/10.1145/276304.276343

[7] Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, Wilton Wu, Aida Zolic, Huibin Shen, Amr Ahmed, Fela Winkelmolen, Miroslav Miladinovic, Cedric Archembeau, Alex Tang, Bhaskar Dutt, Patricia Grao, and Kumar Venkateswar. 2020. Amazon SageMaker Autopilot: A White Box AutoML Solution at Scale. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning* (Portland, OR, USA) *(DEEM'20)*. Association for Computing Machinery, New York, NY, USA, Article 2, 7 pages. https://doi.org/10.1145/3399579.3399870

[8] Sudipto Das, Miroslav Grbic, Igor Ilic, Isidora Jovandic, Andrija Jovanovic, Vivek R. Narasayya, Miodrag Radulovic, Maja Stikic, Gaoxiang Xu, and Surajit Chaudhuri. 2019. Automatically Indexing Millions of Databases in Microsoft Azure SQL Database. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 666–679. https://doi.org/10.1145/3299869.3314035

[9] Vinay Deolalikar and Hernan Laffitte. 2016. Extensive large-scale study of error surfaces in sampling-based distinct value estimators for databases. 1579–1586. https://doi.org/10.1109/BigData.2016.7840767

[10] Marianne Durand and Philippe Flajolet. 2003. Loglog Counting of Large Cardinalities. In *Algorithms - ESA 2003*, Giuseppe Di Battista and Uri Zwick (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 605–617.

[11] Matthias Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. 2015. Efficient and robust automated machine learning. *Advances in Neural Information Processing Systems* 28 (01 2015), 2944–2952.

[12] Philippe Flajolet, Eric Fusy, Olivier Gandouet, and Frédéric Meunier. 2012. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. (03 2012).

[13] Yoav Freund and Robert E. Schapire. 1996. Experiments with a New Boosting Algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning* (Bari, Italy) *(ICML'96)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 148–156.

[14] Phillip B. Gibbons. 2001. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 541–550.

[15] 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 *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB '95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 311–322.

[16] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology* (Genoa, Italy) *(EDBT '13)*. Association for Computing Machinery, New York, NY, USA, 683–692. https://doi.org/10.1145/2452376.2452456

[17] Onur Kocberber, Babak Falsafi, and Boris Grot. 2015. Asynchronous memory access chaining. *Proceedings of the VLDB Endowment* 9, 4 (2015), 252–263. https://doi.org/10.14778/2856318.2856321

[18] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (Nov. 2015), 204–215. https://doi.org/10.14778/2850583.2850594

[19] L.C. Molina, L. Belanche, and A. Nebot. 2002. Feature selection algorithms: a survey and experimental evaluation. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* 306–313. https://doi.org/10.1109/ICDM.2002.1183917

[20] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases* (Seoul, Korea) *(VLDB '06)*. VLDB Endowment, 1049–1058.

[21] City of New York. 2019. NY 311 Service Requests. https://www.kaggle.com/new-york-city/ny-311-service-requests Version 53. Accessed June'21. License: Public Domain.

[22] City of New York. 2019. NY Medallion Vehicles and Drivers. https://www.kaggle.com/new-york-city/ny-medallion-vehicles-and-drivers Version 13. Accessed June'21. License: Public Domain.

[23] City of New York. 2019. NY OATH Hearings Division Case Status. https://www.kaggle.com/new-york-city/ny-oath-hearings-division-case-status Version 13. Accessed June'21. License: Public Domain.

[24] City of New York. 2020. NY Housing Maintenance Code. https://www.kaggle.com/new-york-city/ny-housing-maintenance-code Version 6. Accessed June'21. License: Public Domain.

[25] City of New York. 2020. NYC Parking Tickets. https://www.kaggle.com/new-york-city/nyc-parking-tickets Version 2. Accessed June'21. License: Public Domain.

[26] City of New York. 2021. NYC Fire Incident Dispatch Data. https://www.kaggle.com/new-york-city/nyc-fire-incident-dispatch-data Version 3. Accessed June'21. License: Public Domain.

[27] City of Seattle. 2019. Seattle Checkouts by Title. https://www.kaggle.com/city-of-seattle/seattle-checkouts-by-title Version 4. Accessed June'21. License: Public Domain.

[28] City of Seattle. 2019. Seattle Library Collection Inventory. https://www.kaggle.com/city-of-seattle/seattle-library-collection-inventory Version 7. Accessed June'21. License: Public Domain.

[29] City of Seattle. 2020. Seattle COBAN Logs. https://www.kaggle.com/city-of-seattle/seattle-coban-logs Version 146. Accessed June'21. License: Public Domain.

[30] City of Seattle. 2020. Seattle Road Weather Information Stations. https://www.kaggle.com/city-of-seattle/seattle-road-weather-information-stations Version 144. Accessed June'21. License: Public Domain.

[31] Viswanath Poosala. 1997. *Histogram-Based Estimation Techniques in Database Systems.* Ph.D. Dissertation. USA. UMI Order No. GAX97-16074.

[32] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, Canada) *(SIGMOD '96)*. Association for Computing Machinery, New York, NY, USA, 294–305. https://doi.org/10.1145/233269.233342

[33] A. Shlosser. 1981. On estimation of the size of the dictionary of a long text on the basis of a sample. *Engineering Cybernetics* 19 (01 1981).

[34] Hong Su, Mohamed Zait, Vladimir Barrière, Joseph Torres, and Andre Menck. 2016. Approximate Aggregates in Oracle 12C. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Indianapolis, Indiana, USA) *(CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 1603–1612. https://doi.org/10.1145/2983323.2983353

[35] TPC. 2014. Transaction Processing Performance Council (TPC) TPC Benchmark H (Decision Support) Standard Specification Revision 2.17.1. http://www.tpc.org

[36] Fei Xie, Michael Condict, and Sandip Shete. 2013. Estimating Duplication by Content-Based Sampling *(USENIX ATC'13)*. USENIX Association.

[37] Anatoly Yakovlev, Hesam Fathi Moghadam, Ali Moharrer, Jingxiao Cai, Nikan Chavoshi, Venkatanathan Varadarajan, Sandeep R. Agrawal, Sam Idicula, Tomas Karnagel, Sanjay Jinturkar, and Nipun Agarwal. 2020. Oracle AutoML: A Fast and Predictive AutoML Pipeline. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3166–3180. https://doi.org/10.14778/3415478.3415542

[38] G. K. Zipf. 1932. *Selective Studies and the Principle of Relative Frequency in Language.* Harvard University Press.