A Performance Study of One-dimensional Learned Cardinality Estimation

Yuchen Ji
Osaka University
Osaka, Japan
ji.yuchen@ist.osaka-u.ac.jp

Daichi Amagata
Osaka University, JST Presto
Osaka, Japan
amagata.daichi@ist.osaka-u.ac.jp

Yuya Sasaki
Osaka University, JST Presto
Osaka, Japan
sasaki@ist.osaka-u.ac.jp

Takahiro Hara
Osaka University
Osaka, Japan
hara@ist.osaka-u.ac.jp

ABSTRACT

The latest proximity query processing methods benefit from learned models (indexes) and have shown better performances than non-learned indexing approaches. This paper focuses on one-dimensional data, because querying one-dimensional data is one of the most important operations in database management systems. Specifically, we address the one-dimensional cardinality estimation problem and consider the questions: *Are learned methods suitable for one-dimensional cardinality estimation? If so, how good are they*? To answer these questions, we first design a prototype of a learned method for one-dimensional cardinality estimation. Second, we empirically evaluate the learned method together with existing methods. Then, we analyze the strong and weak points of these methods and find suitable cases for learned methods.

1 INTRODUCTION

Cardinality estimation is a fundamental problem in query optimization. It aims at estimating the number of records required by a query. Most query optimization techniques are cost based [1], and cardinality estimation can help estimate the cost. Recently, *learned* cardinality estimation methods [15, 16] have shown remarkable improvement compared with traditional methods such as histogram-based methods. These learned methods mainly use deep learning models to map given query predicates to the estimated cardinality of query results, because they can handle complex non-linear relationships better than traditional methods [6]. However, existing learned cardinality estimation methods focus on multi-dimensional data [6, 11, 15, 16, 19]. They ignore the most fundamental case, namely one dimension. One-dimensional data and queries play an essential role in database management systems (DBMSs). They support basic operations such as managements of user ids and queries according to timestamps. Cardinality estimation on one dimension can help estimate execution costs and better plan these operations [2]. Investigations on one-dimensional cases can also help develop methods for multi-dimensional cases. For example, traditional histogram-based and sampling-based methods are both extended from one dimension to multiple dimensions.

In the field of machine learning for databases, indexes receive benefits from machine learning [10]. The learned index is first proposed for point and range queries on one-dimensional data [9]. Learned indexes show much better performances compared with traditional indexes, such as B-trees [8]. This observation suggests that the use of a learned model improves cardinality estimation for one-dimensional range queries. To investigate this research question, we make the following contributions.

Designing a learned cardinality estimation model. We design a simple prototype of a learned model for one-dimensional cardinality estimation in Section 3, to compare the learned model with existing techniques.

Empirical evaluations. We conduct experiments by using four real datasets in Section 4, (i) to see the performances of the learned and non-learned cardinality estimation methods and (ii) to confirm whether learned approaches are really promising for one-dimensional cardinality estimation. *Theses two evaluations are the main objective of this paper.* We analyze their strong and weak points. To summarize, there is no clear winner, but the learned method is promising, i.e., its estimation time is the fastest and its error is small enough. Our insights would support implementations/selections of one-dimensional cardinality estimation methods for practical DBMSs.

2 PRELIMINARY

2.1 Problem statement

Consider a dataset D consisting of sorted one-dimensional data with unique integer keys. A range query predicate Q is specified as a central key K and a range R. The query result is the set of records whose keys fall into $[K - R, K + R]$. Here, the goal of cardinality estimation is to estimate the number of records from D satisfying the query predicate. The estimated cardinality is denoted by $\widehat{card(Q)}$ and the real cardinality is denoted by $card(Q)$. (Another equivalent concept of cardinality is *selectivity*, and it represents the percentage of records satisfying the query predicate [20].)

Error metrics. We evaluate Mean Absolute Percentage Error (MAPE) and Q-error to see estimation errors. These errors are widely used for cardinality estimation problems [15, 16, 18] and are respectively defined as:

$$\text{MAPE} = \left| \frac{\widehat{card(Q)} - card(Q)}{card(Q)} \right|$$

and

$$\text{Q-error} = \frac{\max\left(\widehat{card(Q)}, card(Q)\right)}{\min\left(\widehat{card(Q)}, card(Q)\right)}.$$

2.2 Existing methods

Sampling. This is the most straightforward approach to estimating the cardinality of a given range query. The estimated cardinality is obtained by scaling the cardinality on the samples. The performance of sampling is directly related to the number of samples (more samples yields a better accuracy but incurs a higher computational cost).

Histogram. Histogram-based approaches build a histogram on a given dataset and sum up the counts of buckets intersected with a given query predicate as estimated cardinality [4]. Histograms have a long history of being used to solve these estimation tasks [7, 17]. For one-dimensional cases, there are two classical histograms: equal-width and equal-depth. A histogram is essentially a group of linear models, where each bucket is a linear model with a slope of the number of records divided by the width. This *one-level* histogram usually uses a higher computation cost than existing hierarchical learned models [9], because it needs hundreds of add operations to sum up the counts of buckets.

Direct calculation. The cardinality of range queries on one dimensional data can be easily estimated by two point queries because the records in one-dimensional datasets are generally sorted according to keys.

More concretely, we can obtain the exact cardinality after computing the orders of records with $K - R$ and $K + R$ as keys in the dataset for the query Q:

$$card(Q) = Order(K + R) - Order(K - R) + N,$$

where $Order(X)$ indicates the order of the first record whose key is greater than or equal to X in the sorted dataset. N equals 1 if there exists a record whose key equals $K + R$ in the dataset. Otherwise N equals 0. Hence, the computation cost is equivalent to the costs of the two point queries. The computational efficiency can be improved by using indexes.

3 LEARNED MODEL DESIGN

We derive our idea from a learned index structure, Recursive Model Index (RMI) [13]. RMI is the first and state-of-the-art learned index for the *search* problem on one-dimensional data. It stacks learned models to approximate the cumulative distribution function (CDF) and then computes the position of a given key in a sorted array according to the CDF. Here the term "CDF" means the function mapping keys to their corresponding positions in an array.

As shown in Figure 1, the approximated CDF, F, which maps keys to the corresponding positions, can also be used for cardinality estimation. The mapping done by indexes has an intrinsic relationship with the mapping from range queries to estimated cardinality. We can approximate the relationship between cardinality and the range started from K by F':

$$F'(R) = (F(K + R) - F(K - R))/2.$$

RMI usually adopts linear models to approximate F and shows good performance. Therefore, we also employ linear models to approximate F'.

Based on the above observations, we design a prototypic cardinality estimation by refining the original RMI[1]. As shown in Figure 2, this model has a two-level hierarchy. For an input query, the first level directs to a specific model in the second level, and models in the second level then predict the cardinality. We use a

[1]Although we follow the structure of RMI, our model is trained for *estimating the cardinality* of a given query.

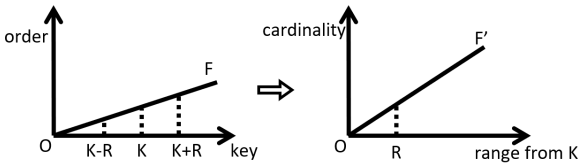

Figure 1: The learned index approximates the CDF to get the orders of the keys. The approximated CDF, *F*, can be used to estimate the cardinality of range queries. Suppose a range query is represented by a central key *K* and range *R*, the cardinality of the range query can be estimated by the orders of keys $(K - R)$ and $(K + R)$. The relationship between cardinality and the range started from the central key *K* can be approximated by *F'*.

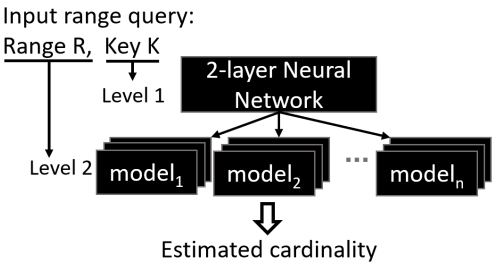

Figure 2: The prototype learned model for one-dimensional cardinality estimation has a two-level hierarchical structure. The first level directs to models in the second level according to a given query. The second level is responsible for cardinality estimation.

two-layer neural network in the first level. In the second level, we organize many multi-scale linear models into an array. Each multi-scale linear model is responsible for a subset of the original dataset.

For the first level, another choice is a linear model with less computation cost than a neural network. However, choosing a linear model will result in a more skewed distribution of subsets for the second level [12], thus negatively affecting performance and training. The neural network is hence a better choice. It is hard for a simple linear model in the second level to cover range queries with totally different scales of selectivity. Therefore, we train some models with different scales for each subset in the second level[2].

During the estimation procedure, the first level takes the central key K of the given query as input and directs it to the corresponding model in the second level. The selected second-layer model estimates the cardinality according to the query range.

From the recent success of learned indexes, we expect that this learned model works well, i.e., it will provide high efficiency and accuracy. We report its practical performance in the next section.

4 EVALUATION

Datasets. We used four real-world datasets with different distributions from the SOSD benchmark [13]. These datasets have recently been used for evaluating learned index works [5, 14]. Each of them contains 200 million 64-bit key/value records:

[2]The selectivity is application dependent. Hence, applications can train this model by specifying the range of selectivity.

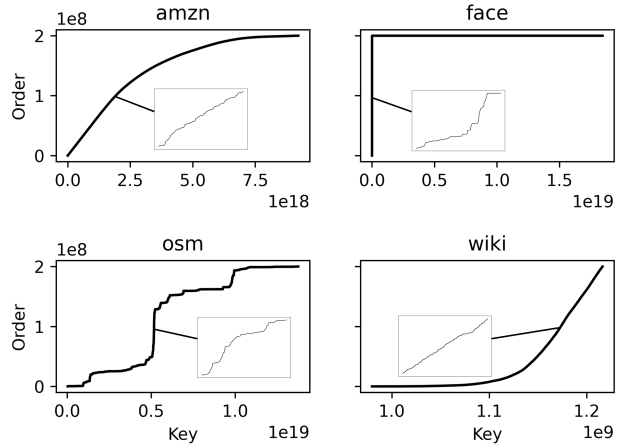

Figure 3: CDFs of evaluation datasets. Zoomed regions show plots of 200 consecutive keys.

Table 1: Overview of evaluated methods

Method	Description
LC	Learned model designed in Section 3
SA2	Sampling $1 * 10^{-2}$ records of original dataset
SA3	Sampling $1 * 10^{-3}$ records of original dataset
SA4	Sampling $1 * 10^{-4}$ records of original dataset
HM	An equal-depth histogram having $1 * 10^5$ buckets
DC	Direct calculation using RMI as the index

- amzn: book popularity data from Amazon.
- face: Facebook user IDs sampled randomly.
- osm: cell IDs from Open Street Map.
- wiki: timestamps from Wikipedia.

Figure 3 plots CDFs of the four datasets. Zoomed regions show small segments of datasets. The face dataset contains tens of outliers with much larger keys than the others.

Environment. All experiments were conducted on a server with an Intel Xeon Gold 6254 @3.10GHz CPU and 768GB RAM, running Ubuntu 18.04 LTS. The evaluated methods were implemented in Python.

Model setting. We evaluated the methods listed in Table 1. In the case of the sampling methods, we estimated the cardinality based on calculated cardinality of sampled datasets by binary search. An equal-depth histogram has the same number of records in each bucket. For the direct calculation method, we built an RMI index on the original dataset. Compared with classical B-Trees [3], RMI shows better computational and space costs [13]. Recall that direct calculation returns the exact cardinality, so we measured the computation time and model size for direct calculation. For the learned method, we used a two-layer neural network with 16 neurons in the hidden layer for the first level. We set the number of models in the second level to $1 * 10^5$ to leverage the storage cost and accuracy. Table 2 shows the model sizes of all methods. We controlled the sizes (space consumptions) of LC, HM, SA3, and DC so that they were (almost) the same.

Training of LC. For the first level, we assume that a given dataset is equally divided into subsets for models of the second level. We composed keys and corresponding ids of models into pairs as the training data. After the training of the first level was done, we partitioned the original dataset according to the output of the first level. For the second level, we trained the group of models sequentially. We prepared training data of multiple scales for each model. For each scale, we generated 1000 query predicates (K, R) and calculated the true cardinality as the training data.

Table 2: Model sizes of evaluated methods (MB)

Method	LC	HM	SA2	SA3	SA4	DC
Model size	1.6	0.8	16	1.6	0.16	1.6

The keys and ids were both scaled to a maximum of 100 during the training and evaluation. To train the neural network of the first level, we used Adam as the optimizer. We set the learning rate to $1 * 10^{-4}$. It took two epochs to finish the training. The loss remained large when the network was trained on osm, because a two-layer neural network cannot approximate the skewed distribution well. We divided the datasets into subsets for models in the second level according to the predicted results of the trained neural network. Therefore, the high loss of the neural network did not matter. We fitted the linear models of the second level to the training data by using non-linear least squares. In the case of the face dataset, there were tens of outliers that made it hard for training. We removed the outliers when we scaled the keys for model training.

Workloads. For query predicates (K, R), first, we selected scales of selectivity[3] for query range R from $\{10^{-5}, 10^{-4}, 10^{-3}\}$. Then we randomly generated 1000 pairs of (K, R) for each scale of selectivity. The keys generated are limited within the range of existing keys in the datasets. The generated R was floated around the selected selectivity with possible magnitudes within $(10^{-1}, 10^{1})$.

Evaluation results. Table 3 shows the estimation errors on all datasets and queries with a selectivity of around 10^{-4}. The 50th/95th/99th values show the corresponding percentile errors. Table 4 shows estimation errors on the wiki dataset with selectivities of around 10^{-5}, 10^{-4}, and 10^{-3}.

The MAPE errors and Q-errors do not show significant differences. Most MAPE errors are much smaller than 1, with zeros after their decimal points. According to the definitions, the similarity of MAPE errors and Q-errors is easy to understand. However, for SA4, whose selectivity of sampling hardly matches the queries' selectivity, its Q-errors are extremely large.

Exp-1 (LC vs. SA). Sampling methods can easily find a trade-off between accuracy and storage cost. According to the errors shown in Tables 3 and 4, we study the following: (i) Generally, LC in the current setting has a competitive accuracy over SA3. (ii) The 50th errors of LC are usually slightly larger than those of SA3, whereas LC has smaller 95th and 99th errors. This means that the distribution of LC's errors is more even than that of SA3. (iii) Sampling methods have larger errors for smaller selectivity. In contrast, the variation of LC's errors for different scales of selectivity is more stable.

The MAPE errors of the three sampling methods are linear to their sampling scales. This stable multiple of errors for sampling methods of continuous scales makes sampling methods suitable for a performance baseline. Sampling methods with different scales can be treated as a trade-off between accuracy and storage cost. It is always easy for sampling methods to reduce errors.

[3]We omit the term "1∗".

Table 3: Estimation errors for cardinality estimation using queries with a selectivity of around 10^{-4}.

Dataset	amzn						face					
Metric	MAPE			Q-error			MAPE			Q-error		
	50th	95th	99th	50th	95th	99th	50th	95th	99th	50th	95th	99th
LC	$5.5*10^{-3}$	$1.2*10^{-1}$	$3.3*10^{-1}$	1.0055	1.13	1.34	$3.2*10^{-2}$	$9.0*10^{-2}$	$3.6*10^{-2}$	1.032	1.097	1.14
HM	$6.3*10^{-3}$	$1.0*10^{-1}$	$2.0*10^{-1}$	1.0063	1.11	1.22	$1.4*10^{-2}$	$4.4*10^{-2}$	$6.2*10^{-2}$	1.014	1.045	1.064
SA2	$7.8*10^{-3}$	$4.5*10^{-2}$	$8.2*10^{-2}$	1.0078	1.045	1.082	$8.9*10^{-3}$	$1.3*10^{-2}$	$1.6*10^{-2}$	1.0089	1.014	1.016
SA3	$8.2*10^{-2}$	$3.8*10^{-1}$	1.0	1.087	1.46	$3.1*10^{2}$	$9.0*10^{2}$	$1.4*10^{-1}$	$1.6*10^{-1}$	1.098	1.16	1.19
SA4	1.0	2.6	6.1	$1.4*10^{3}$	$5.4*10^{3}$	$5.4*10^{3}$	1.0	1.5	1.6	$1.9*10^{4}$	$2.5*10^{4}$	$2.7*10^{4}$
Dataset	osm						wiki					
Metric	MAPE			Q-error			MAPE			Q-error		
	50th	95th	99th	50th	95th	99th	50th	95th	99th	50th	95th	99th
LC	$1.1*10^{-1}$	$3.4*10^{-1}$	$8.9*10^{-1}$	1.11	1.37	1.96	$6.5*10^{-3}$	$3.0*10^{-2}$	$1.2*10^{-1}$	1.0065	1.031	1.12
HM	$1.2*10^{-3}$	$5.6*10^{-2}$	$3.2*10^{-1}$	1.0012	1.058	1.41	$1.0*10^{-3}$	$1.0*10^{-2}$	$7.0*10^{-2}$	1.0010	1.010	1.072
SA2	$1.1*10^{-3}$	$2.8*10^{-2}$	$1.3*10^{-1}$	1.0011	1.029	1.14	$1.2*10^{-3}$	$6.8*10^{-3}$	$2.4*10^{-2}$	1.0013	1.0068	1.024
SA3	$1.1*10^{-2}$	$2.8*10^{-1}$	1.0	1.011	1.33	6.54	$1.2*10^{-2}$	$7.1*10^{-2}$	$2.6*10^{-1}$	1.012	1.072	1.32
SA4	$1.1*10^{-1}$	2.4	$1.3*10^{1}$	1.12	10.88	$2.3*10^{4}$	$1.2*10^{-1}$	1.0	1.34	1.12	$4.5*10^{2}$	$5.9*10^{3}$

Table 4: Estimation error comparison at different scales of range query using the wiki dataset.

Metric	MAPE								
	50th			95th			99th		
Selectivity	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}
LC	$2.7*10^{-2}$	$6.5*10^{-3}$	$1.2*10^{-2}$	$5.4*10^{-2}$	$3.0*10^{-2}$	$8.7*10^{-2}$	$6.1*10^{-2}$	$1.2*10^{-1}$	$3.0*10^{-1}$
HM	$1.1*10^{-4}$	$1.0*10^{-3}$	$1.0*10^{-3}$	$1.1*10^{-3}$	$1.0*10^{-2}$	$7.1*10^{-2}$	$8.4*10^{-3}$	$7.0*10^{-2}$	$2.1*10^{-1}$
SA2	$1.3*10^{-4}$	$1.2*10^{-3}$	$1.2*10^{-2}$	$6.6*10^{-4}$	$6.8*10^{-3}$	$6.7*10^{-2}$	$2.6*10^{-3}$	$2.4*10^{-2}$	$2.7*10^{-1}$
SA3	$1.3*10^{-3}$	$1.2*10^{-2}$	$1.2*10^{-1}$	$6.5*10^{-3}$	$7.1*10^{-2}$	1.0	$2.7*10^{-2}$	$2.6*10^{-1}$	1.4
SA4	$1.3*10^{-2}$	$1.2*10^{-1}$	1.0	$6.6*10^{-2}$	1.0	3.6	$2.2*10^{-1}$	1.3	6.1
Metric	Q-error								
	50th			95th			99th		
Selectivity	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}
LC	1.03	1.0065	1.012	1.054	1.031	1.09	1.06	1.12	1.35
HM	1.00011	1.0010	1.0098	1.0011	1.010	1.075	1.0085	1.072	1.24
SA2	1.00013	1.0013	1.012	1.00066	1.0068	1.068	1.0026	1.024	1.32
SA3	1.0013	1.012	1.13	1.0065	1.072	33	1.027	1.32	$5.7*10^{2}$
SA4	1.013	1.12	$1.9*10^{3}$	1.068	$4.5*10^{2}$	$3.8*10^{3}$	1.25	$5.9*10^{3}$	$4.4*10^{3}$

Exp-2 (LC vs. HM). Tables 3 and 4 show that HM generally returns smaller errors than LC. Similar to SA, HM shows degraded performance with smaller selectivity of queries. HM has a more significant advantage over LC for queries with a selectivity of 10^{-3}.

Exp-3 (Performances on osm). LC has the worst accuracy on osm among all datasets, whereas the other methods do not show decreased performances on osm. This is mainly attributed to the distribution of osm. Figure 3 shows that osm is the most skewed. The small segment shown in the zoomed region of osm shows that the distribution lacks local structures, making it difficult for LC to learn the CDF. Similar situations are met in indexing works [13]. HM and the sampling methods are not bothered a lot by the distribution of osm, because they do not need to learn the distribution.

Exp-4 (Performances on amzn, face, and wiki). The CDFs of amzn and wiki show very smooth distributions. The zoomed small segments of amzn and wiki also look similar. Considering zoomed segments of face, face also lacks local structures, similarly to osm. As a result, LC has more significant errors on face compared with its errors on amzn and wiki.

Exp-5 (Estimation time). Table 5 shows the average estimation times on different datasets with a selectivity of 10^{-4}. LC is much faster than the other methods. DC takes less time than HM and SA4 on amzn, face, and wiki. When RMI is used as the index, the time cost of a point query consists of the inference time (evaluated by RMI) and the search time (for searching around the predicted order). As DC uses two point queries to estimate cardinality, it is reasonable that DC needs longer time than LC. We see that LC, HM, and SA have stable times on different datasets. DC needs more time on osm than the other datasets. The point queries on osm incur a long search time [13]. HM scans the border values of buckets, and this scanning contributes to a major part of the time costs. The sampling methods with large samples involve many record accesses, thereby SA2 and SA3 are very slow.

5 DISCUSSION

The main advantage of the prototype learned method LC is its computational efficiency. This comes from its structure, which is based on simple learned models. At the same time, the estimation errors of LC seem a bit larger than those of the other evaluated methods. In this section we summarize and discuss our findings.

Table 5: Estimation time comparison for different methods on different datasets (microseconds)

Dataset	LC	HM	SA2	SA3	SA4	DC
amzn	15.9	139.2	44821,5	1176.6	129.8	77.6
wiki	15.9	145.0	42313.6	1162.3	129.9	91.8
face	16.9	148.4	44743.2	1230.1	138.5	93.0
osm	17.6	153.4	47847.8	1271.0	143.1	196.6

Best performance. LC has the best computational efficiency. Its error is a bit larger than those of the other models but is absolutely small. DC provides the true value of cardinality, showing the best accuracy. At the same time, its estimation time normally outperforms those of the sampling methods and HM. Consequently, LC and DC are the options, but for *estimation* purpose, we consider that LC is better suited, as it yields high efficiency and small error.

Differences among LC, HM, and DC. As mentioned in Section 2.1, HM is a group of linear models. LC, HM, and DC with RMI all make use of models. The model differences lead to differences in their performances. DC calculates cardinality by finding the orders of two border keys of the query range. It involves twice the computation cost of RMI. Compared with DC, LC infers the difference of the orders of two border keys instead of inferring the orders. Hence, LC reduces the computation cost to one pass of the learned models. Models of HM take charge of counting their buckets. When the query range covers multiple buckets, which is a typical case, HM accumulates counts of all covered buckets. Only on the border buckets intersected by the query range, inferences by models need to be executed. The errors in HM come from estimations of the border buckets.

Further optimization. If we treat LC as a baseline, better methods for one-dimensional cardinality estimation should have smaller estimation errors and competitive computational efficiency. A possible way is to add some extra models to LC to get more accurate estimations. However, complex models are not suitable because they incur high time costs. Therefore, the issue is how to make LC more accurate while retaining simple structures. If we seek to optimize models compared with DC, then we need to cut down its computation time. With RMI as an index, DC consumes about 20 multiplication operations (varying among the different models used in RMI) and tens of add operations. Less computation than DC means that only several multiplication operations are allowed. This is a new research challenge.

Training cost. For learned methods, we need to consider the training costs when bringing them to applications. The training cost of LC consists of two parts. (i) For model training, LC shares similar costs with the original RMI model, which DC uses. In our evaluation, to train the prototype learned model on a dataset consisting of 200 million records, it took about a half hour for a simple neural network and 10 minutes for $1*10^5$ linear models. (ii) For the preparation of training data, LC has two ways, passive and active. The query results produced by DBMSs can be utilized for training. However, this passive way cannot guarantee sufficient amounts of data to train the models. Besides, active collection of training data for learned models can be expensive [6]. LC needs a longer time than DC to collect training data, because DC does not need to obtain true cardinality. However, both training data collection and model training are done *once*, so the offline cost of LC would not be a drawback in practice.

Summary. To summarize our discussion, LC and DC are options of one-dimensional cardinality estimation if their training costs are not bottlenecks for applications. They have accuracy-time trade-off. We provided the directions of further optimizations for LC and DC.

6 CONCLUSION

In this study, we addressed an unanswered question: *Are learned methods suitable for one-dimensional cardinality estimation?* We designed a prototype learned structure for one-dimensional cardinality estimation. We empirically evaluated the performances of the learned method and existing methods to investigate their advantages and disadvantages. The evaluation results answer the question: A learned method based on the current state-of-the-art model is useful for quick and accurate cardinality estimation. In future work, we will seek to optimize the prototypic learned method to achieve better accuracy by using other structures. We also plan to design a strategy to collect training data, e.g., to judge whether collected training data satisfy the needs.

ACKNOWLEDGMENTS

This research is partially supported by JST PRESTO Grant Number JPMJPR1931 and JST CREST Grant Number JPMJCR21F2.

REFERENCES

[1] Nicolas Bruno and Surajit Chaudhuri. 2002. Exploiting statistics on query expressions for optimization. In *SIGMOD*. 263–274.
[2] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 34–43.
[3] Douglas Comer. 1979. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)* 11, 2 (1979), 121–137.
[4] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends Databases* 4, 1–3 (2012), 1–294.
[5] Andrew Crotty. 2021. Hist-Tree: Those Who Ignore It Are Doomed to Learn.. In *CIDR*.
[6] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *SIGMOD*. 1035–1050.
[7] Yannis Ioannidis. 2003. The History of Histograms (Abridged) *(VLDB)*. 19–30.
[8] Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020. RadixSpline: A Single-Pass Learned Index. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM)*. Article 5.
[9] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *SIGMOD*. 489–504.
[10] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. AI Meets Database: AI4DB and DB4AI. In *SIGMOD*. 2859–2866.
[11] Qiyu Liu, Yanyan Shen, and Lei Chen. 2021. LHist: Towards Learning Multi-dimensional Histogram for Massive Spatial Data. In *ICDE*. 1188–1199.
[12] Marcel Maltry and Jens Dittrich. 2021. A Critical Analysis of Recursive Model Indexes. *arXiv preprint arXiv:2106.16166* (2021).
[13] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. 2020. Benchmarking Learned Indexes. *PVLDB* 14, 1 (2020), 1–13.
[14] Mihail Stoian, Andreas Kipf, Ryan Marcus, and Tim Kraska. 2021. PLEX: Towards Practical Learned Indexing. In *International Workshop on Applied AI for Database Systems and Applications (AIDB)*.
[15] Ji Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation for Similarity Queries. In *SIGMOD*. 1745–1757.
[16] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, Rui Zhang, and Yoshiharu Ishikawa. 2021. Consistent and flexible selectivity estimation for high-dimensional data. In *SIGMOD*. 2319–2327.
[17] Edward J Wegman. 1969. *Nonparametric probability density estimation.* Technical Report. North Carolina State University. Dept. of Statistics.
[18] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from Both Data and Queries for Cardinality Estimation. In *SIGMOD*. 2009–2022.
[19] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *PVLDB* 13, 3 (2019), 279–292.
[20] Xiaohui Yu and Ada Fu. 2001. Piecewise Linear Histograms for Selectivity Estimation. In *International Symposium on Information Systems and Engineering (ISE)*. 319–326.