Monotonic Cardinality Estimation for Similarity Selection: A Deep Learning Approach

Yaoshu Wang
UNSW
Australia
yaoshuw@cse.unsw.edu.au

Chuan Xiao
Nagoya University
Japan
chuanx@nagoya-u.jp

Jianbin Qin
The University of Edinburgh
United Kingdom
jqin@inf.ed.ac.uk

Xin Cao
UNSW
Australia
xin.cao@unsw.edu.au

Yifang Sun
UNSW
Australia
yifangs@cse.unsw.edu.au

Wei Wang
UNSW
Australia
weiw@cse.unsw.edu.au

ABSTRACT

We explore the following problem: Given a set of records $\mathcal{D}$, a query record x, a distance function and a threshold θ, estimate the cardinality, i.e., the number of records in $\mathcal{D}$ whose distances to x are no greater than θ. Answering this problem accurately and efficiently is essential to many database systems that rely on the estimated cardinality values for query optimization or data exploration. Despite quite a few methods proposed for this problem, they either deliver inferior performance or only apply to some specific distance functions. Furthermore, most existing methods do not guarantee that the estimated cardinality is monotonic with respect to the threshold, and thus the results are inconsistent and lack interpretability for applications like data exploration. In this paper, we propose a novel and generic method that can be appllied to any data type and distance function for cardinality estimation of similarity selection. Based on deep neural networks, our method consists of a feature extraction model and a regression model. The feature extraction model converts original data and threshold to a Hamming space, in which the encoder-decoder-based regression model exploits the nature of cardinality to achieve both accuracy and monotonicity. We develop specific training strategy as well as techniques for fast online estimation. The effectiveness and the efficiency of our method are demonstrated through extensive experiments on real datasets.

PVLDB Reference Format:
Yaoshu Wang, Chuan Xiao, Jianbin Qin, Xin Cao, Yifang Sun, and Wei Wang. Monotonic Cardinality Estimation for Similarity Selection: A Deep Learning Approach. *PVLDB*, 12(x): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/3275536.3275539

1. INTRODUCTION

Proceedings of the VLDB Endowment, Vol. 12, No. x
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3275536.3275539

Cardinality estimation is an essential procedure in many database systems that rely on the estimated cardinality values for query optimization or data exploration. In this paper, we study the cardinality estimation for similarity selection queries, i.e., given a set of records $\mathcal{D}$, a query record x, a distance function and a threshold θ, estimate the the number of records in $\mathcal{D}$ whose distances to x are no greater than θ. This problem is important for many applications, such as image retrieval, near-duplicate detection, and entity resolution. For example, in image retrieval, images are converted to binary vectors and the vectors whose Hamming distances to the query are within a threshold of 16 are identified for further image-level verification [63]. Estimating cardinalities of similarity selection helps compute the cost of operations and the execution orders of query plans that involve a similarity predicate. In data exploration, fast and accurate cardinality estimation for similarity selection is necessary to explore the properties of data for tasks that involve similarities, e.g., how many images in the dataset may pertain to the same person as the one in the query.

State-of-the-art methods for cardinality estimation of similarity selection can be divided into the following categories: sampling [58,60], auxiliary index structures [30,38,47], traditional (non-deep) learning [28,46], and deep learning [35,52] methods. Although these methods exhibit advantages in some aspects, they fail to address the major technical issues of cardinality estimation for similarity selection. First, the method should be accurate and fast. E.g., sampling and traditional learning (kernel-based) methods need a large set of samples for accuracy, and thus become either slow or inaccurate when applied on large datasets. Second, the method should be able to support a variety of data types and functions. However, some methods only target one or few distance/similarity functions, e.g., Jaccard similarity [38] and Levenshtein distance [30]. In addition, in applications like data exploration, users may want the estimated cardinality to be consistent and interpretable. E.g., the exact value of the cardinality is monotonically increasing with the threshold. When the user inputs a greater threshold, a larger (or equal) number of results is preferable, so the user can interpret the effect of the change of threshold for better analysis of data. Despite a few deep learning models that guarantees the monotonicity [19,61], they are not directly targeting the cardinality estimation, and thus the performance is rather

bad when applied to this problem.

Seeing these challenges, we propose a novel method to tackle the cardinality estimation problem for similarity selection. Our method is not only accurate and fast but also designed in a generic fashion. In addition, unlike existing methods (except the kernel-based method [46]) for this problem, our method guarantees the monotonicity of cardinality with respect to the threshold and thus yields more interpretability of the estimated results.

To achieve the above goals, we propose a method that separates data modelling and cardinality estimation into two components: a *feature extraction* component that maps original data and distance threshold to a Hamming space, and a *regression* component that models the estimation as a regression problem and estimates the cardinality using the converted vectors and threshold in the Hamming space. By the feature extraction, our method can be applied to *any* data type and distance/simiarity function. The regression component employs deep neural networks which have been shown competitive for regression problems by recent studies from the machine learning research community. Specifially, it is devised based on an *encoder-decoder* model for *incremental* predictions, thus to exploit the incremental property of cardinality (i.e., when the threshold is increasing, the increment of cardinality only originates from the results in the increased range of distance) for both accuracy and monotonicity. To learn the incremental predictions for different query records and thresholds, we design a loss function and a dynamic training strategy, both specific to our regression model, to focus on dealing with the distance values that tend to cause more inaccuracy, so that the method can generalize well through training. Finally, optimization techniques are developed for fast online estimation. Experiments are carried out on four distance functions over ten real datasets. The results demonstrate that our method is more accurate than existing methods and runs much faster with a moderate model size. The experiments also compare the monotonicity performance of various methods and show 100% of monotonicity of our method on real datasets.

Our contributions are summarized as:

1. We develop an accurate and fast method for the cardinality estimation of similarity selection queries. The proposed method gurantees the monotonicity of cardinality with respect to the input threshold.
2. Through feature extraction and regression, our method is generic to any input data type and distance function, and exploits the incremental property of cardinality to achieve accuracy and monotonicity.
3. We design training techniques that favor our method as well as acceleration techniques for online estimation.
4. We conduct extensive experimental evaluation to demonstrate the superiority of the proposed method over existing ones.

The rest of the paper is organized as follows. Section 2 defines the problem and reviews related work. Section 3 presents the overall framework of our method. Section 4 discusses case studies for feature extraction. Section 5 presents the detailed model design of regression. The model training is covered by Section 6. Section 7 is dedicated to model acceleration for online estimation. Experimental results are reported in Section 8. Section 9 concludes the paper.

2. PRELIMINARIES

Table 1: Frequently Used Notations

Symbol	Definition
$\mathcal{O}$	a record universe
$\mathcal{D}$	a dataset
x, y	records in $\mathcal{O}$
f	a distance function
θ, $\theta_{\max}$	a distance threshold and its maximum value
c, $\widehat{c}$	cardinality and the estimated value
g	the regression function
h	the feature extraction function
$\mathbf{x}$	the binary representation of x
d	the dimensionality of $\mathbf{x}$
τ, $\tau_{\max}$	a threshold in Hamming space and its maximum value
$\mathbf{e}^i$	the embedding of distance i
$\mathbf{z}_x^i$	the embedding of $\mathbf{x}$ and distance i

2.1 Problem Definition and Notations

Let $\mathcal{O}$ be a universe of records. x and y are two records in $\mathcal{O}$. $f : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ is a distance (similarity) function which evaluates the distance (similarity) of a pair of records. Common distance (similarity) functions include Hamming distance, Jaccard similarity, Levenshtein (edit) distance, Euclidean distance, etc. Without loss of generality, we assume f is distance function. Given a collection of records $\mathcal{D} \subseteq \mathcal{O}$, a query record $x \in \mathcal{O}$, and a threshold θ, a similarity selection is to find all the records $y \in D$ such that $f(x, y) \leq \theta$. We formally define the problem of cardinality estimation for similarity selection.

PROBLEM 1. *Given a collection $\mathcal{D}$ of records, a query record x, a distance function f, and a threshold θ, the cardinality estimation for similarity selection is to estimate the result number of the similarity selection, i.e., $|\{\, y \mid f(x, y) \leq \theta, y \in \mathcal{D} \,\}|$.*

Mean absolute percentage error (MAPE), also called mean relative error, and mean squared error (MSE) are two widely used evaluation metrics in the cardinality estimation problem [26, 43, 46, 60]. We adopt these two metrics to evaluate the accuracy of cardinality estimation in this paper.

Table 1 lists the frequently used notations in this paper. We use bold uppercase letters, e.g., $\mathbf{A}$ for matrices; bold lowercase letters, e.g., $\mathbf{a}$, for vectors; and non-bold lowercase letters for scalars, e.g., a or other variables. Uppercase Greek symbols (e.g., Ψ) are used to denote neural networks. $\mathbf{A}[i, *]$ and $\mathbf{A}[*, i]$ denote the i-th row and the i-th column of $\mathbf{A}$, respectively. $\mathbf{a}[i]$ denotes the i-th dimension of $\mathbf{a}$. Semicolon represents the concatenation of vectors. E.g., given an a-dimensional vector $\mathbf{a}$ and a b-dimensional vector $\mathbf{b}$, $\mathbf{c} = [\mathbf{a}; \mathbf{b}]$ means that $\mathbf{c}[1\,..\,a] = \mathbf{a}$ and $\mathbf{c}[a+1\,..\,a+b] = \mathbf{b}$. Colon represents the construction of a matrix by column vectors (matrices). E.g., $\mathbf{C} = [\mathbf{a} : \mathbf{b}]$ means that $\mathbf{C}[*, 1] = \mathbf{a}$ and $\mathbf{C}[*, 2] = \mathbf{b}$.

2.2 Prior Works

Cardinality estimation has many facets, such as distinct values counting [25], join size estimation [32, 56], query cardinality estimation of rectangle range [12, 26]. We mainly summarize the related work on query cardinality estimation with distance (similarity) functions.

2.2.1 Database Methods

Auxiliary structures and sampling methods are two main categories. Auxiliary structures are index structures or hash buckets that pre-compute the statistic information of a dataset

and accelerate the estimation procedure. The histogram method [27] splits a domain into several buckets, and save the number records in each bucket. However, there is no valid criterion to create buckets [46]. For binary vectors, histograms [50] are constructed to count the number of records by partitioning dimensions into buckets and enumerating binary vectors and thresholds. For strings and sets, OptEQ [37] and LC [38] utilize semi-lattice structures to store possible situations to transfer one object to other, but they have their own weaknesses, such that OptEQ only adapts to very small edit distance thresholds and relatively long strings, and LC can only solve low-dimensional vectors generated by minhash [41]. SEPIA [30] and VSol [47] adopt inverted indexes for estimation, but the drawback is high estimation cost and low accuracy. Another line of work is sampling methods, e.g., uniform sampling, adaptive sampling [42], and sequential sampling [24]. State-of-the-art sampling strategies [29, 40, 59, 64] focus on join size estimation in query optimization, and is difficult to be adopted in our query estimation problem with distance functions. Sampling methods often combine strategies of locality sensitive hashing (LSH) [22, 39] to improve their performance, but the number of LSH functions should be not large. These methods do not scale well for high-dimensional data, because it is hard to achieve high accuracy with a small set of samples. A state-of-the-art method was proposed in [60] for high dimensions. The idea is to construct multiple hash buckets and adopt importance sampling.

2.2.2 Traditional Machine Learning Models

The curse of dimensionality is a major challenge to traditional ML models. The kernel-based estimator [46] maps high-dimensional data to a one-dimensional metric space. However, particular kernel functions cannot fully describe the complex distribution of metric space. Traditional regression models [28], such as support vector regression, logistic regression, and gradient boosting tree, were also explored to solve the cardinality estimation problem. A query-driven approach [13] was proposed to learn several query prototypes (i.e., interpolation) that are differentiable. However, it is infeasible for data with high dimensions due to too many prototypes that should be constructed.

2.2.3 Deep Regression Models

Recently, deep regression models have been explored for various computer vision tasks, e.g., facial landmark detection [55], age estimation [53], and head pose estimation [44]. These models are mainly based on convolutional neural network and replace the classification softmax layer with a fully connected regression layer. Most of them focus on extracting useful features from images but not consider the problem of large range labels. In query optimization, deep learning models are recently adopted to learn the best join order [36, 45] or estimating the join size [34]. Deep reinforcement learning is also explored to generate query plans [48]. However, these models cannot be directly adopted to solve our problem with similarity selection. Some deep regression models can be adapted to cardinality estimation. The mixture of expert model (MoE) [52] targets various inputs. It utilizes a sparsely-gated mixture-of-experts layer and assigns good experts (models) to these inputs. Besides, based on a hierarchical deep regression model, the recursive-model index [35] (RMI) is a learned index structure that can be used to replace the traditional B-tree index for range queries.

2.2.4 Deep Monotonic Models

For monotonic cardinality estimation, existing monotonic models are mainly two categories: fully monotonic models where all features are monotonic to predictions and partial monotonic models where only partial features are monotonic. In our problem, because there is only one monotonic feature (the threshold), we focus on partial monotonic models. MonoNet [19] is a min-max 4-layer deep neural network that allows partially monotonic features. Lattice [21,23,61] is a kind of monotonic models that adopt lattice structure to construct all combinations of monotonic interpolation values. To handle high-dimensional monotonic features, ensemble of lattices [21] splits lattices into several small pieces using ensemble learning. A partially monotonic deep network (DLN) was proposed in [61]. It consists of multiple calibration layers and ensemble of lattices layers. Calibrations and lattices keep the monotonicity and multiple such layers make the model deeper and more flexible.

3. CARDINALITY ESTIMATION FRAMEWORK

3.1 Basic Idea

Let $c(x, \theta)$ denote the cardinality for a query x with threshold θ. We model this as a regression problem with a unique framework designed to alleviate the challenges mentioned in Section 1. We would like to find a function $\widehat{c}$ within a function family, such that $\widehat{c}$ returns similar values as c for any valid input x, i.e., $\widehat{c}(x, \theta) \approx c(x, \theta), \forall x$ and θ. We consider $\widehat{c}$ that belongs to the following family: $\widehat{c} \coloneqq g \circ h$, where $h(x, \theta) = (\mathbf{x}, \tau)$, $g(\mathbf{x}, \tau) \in \mathbb{Z}_{\geq 0}$, $\mathbf{x} \in \{0, 1\}^d$, and $\tau \in \mathbb{Z}_{\geq 0}$. Intuitively, we can deem h as a *feature extraction* function, which maps an object x and a threshold θ to a fixed-dimensional binary vector $\mathbf{x}$ and an integer threshold τ. Then, the function g essentially performs the regression using the transformed input (i.e., the $(\mathbf{x}, \tau)$ pair) and the ground truth value c. The rationales of such a design are the following:

- This design separates data modelling and cardinality estimation into two functions, h and g, respectively. On one hand, this allows the system to cater for different data types, and distance functions. On the other hand, it allows us to choose the best machine learning models and techniques for the estimation problem. To decouple the two components, some common interface needs to be established, and that is exactly why we pose the constraints that (i) $\mathbf{x}$ belonging to a Hamming space, and (ii) τ is an non-negative integer. We will leave other interface design choices to future work due to their added complexity. For example, it is entirely possible to restrict $\mathbf{x}$ to $\mathbb{R}^d$ and $\tau \in \mathbb{R}$. While this will definitely increase the modelling power of the framework, this will inevitably result in more complex models that are potentially difficult to train.

- The design can also be deemed as an instance of the *encoder-decoder* framework, where two functions h and g are used for some prediction tasks. As a close analog, Google translation [31] trains an h that maps inputs in the source language to a latent representation, and then train a g that maps the latent representation to the destination

language. As such, it can support translation between m languages by training only $2m$ functions, instead of $m(m+1)$ direct translation functions.

Our function $\widehat{c} = g \circ h(x, \theta)$ is monotonic w.r.t. θ, provided that both feature extraction function h and regression model g are monotonic, as stated by the following lemma.

LEMMA 1. *Consider a function $h(x, \theta)$ monotonically increasing with θ and a regression model $g(\mathbf{x}, \tau)$ monotonically increasing with τ, our framework $g \circ h(x, \theta)$ is monotonically increasing with θ.*

PROOF. Consider two queries (x, θ_1) and (x, θ_2), where $\theta_1 \leq \theta_2$. $(\mathbf{x}, \tau_1) = h(x, \theta_1)$ and $(\mathbf{x}, \tau_2) = h(x, \theta_2)$. Because $h(x, \theta)$ is monotonically increasing with θ, $\tau_1 \leq \tau_2$. Because $g(\mathbf{x}, \tau)$ is monotonically increasing with τ, $g(\mathbf{x}, \tau_1) \leq g(\mathbf{x}, \tau_2)$. Because $g \circ h(x, \theta_1) = g(\mathbf{x}, \tau_1)$ and $g \circ h(x, \theta_2) = g(\mathbf{x}, \tau_2)$, $g \circ h(x, \theta_1) \leq g \circ h(x, \theta_2)$. Hence the monotonicity is proved. □

We choose deep models for the regression g. First, deep models with moderate sizes are able to infer the cardinality very fast, and can be accelerated by modern hardware that optimizes batch strategies or matrix manipulation. Second, although deep models usually require large training datasets for good performance, the training data for our problem can be easily and efficiently acquired by running state-of-the-art similarity search algorithms on the dataset.

3.2 Feature Extraction

The process of feature extraction is to transfer any data type and distance function into binary representation and discrete threshold. Formally, we have a function $h_{rec} : \mathcal{O} \to \{0, 1\}^d$; i.e., given any record $x \in \mathcal{O}$, h_{rec} maps x to a d-dimensional binary vector, denoted by $\mathbf{x}$, called x's binary representation. We can plug in any user-defined functions or neural networks for feature extraction. For the sake of estimation accuracy, the general criteria is that the distance (e.g., Hamming distance) of the target d-dimensional binary vectors can equivalently or approximately capture the semantics of the original distance function. We will show some example feature extraction methods and a series of case studies in Section 4.

Besides the conversion to binary representations, we also have a monotonically increasing (as demanded by Lemma 1) function $h_{thr} : [0, \theta_{\max}] \to [0 \,..\, \tau_{\max}]$ to convert the threshold. $\theta_{\max}$ is a parameter for the maximum threshold (reasonably large for the similarity selection to make sense) we are going to support for the distance function f, and $\tau_{\max}$ is a tunable parameter to control the model size. Given $\theta \in [0, \theta_{\max}]$, h_{thr} maps it to an *integer* between 0 and $\tau_{\max}$, denoted by τ. The purpose of such conversion is: for real-valued distance functions, it makes the distances countable; for integer-valued distance functions, it can reduce the threshold to a small number, hence to prevent the model growing too big when the input threshold is large. In doing so, we are able to use $(\tau + 1)$ estimators to predict the cardinalities in $[0, 1]$, $[1, 2]$, ..., and $[\tau - 1, \tau]$, respectively. The design of threshold mapping depends on how original data are converted to binary representations. In general, a mapping with less skew leads to better performance. Using the threshold of the Hamming distance between binary representations is not necessary, but would be a preferable option. A few case studies will be given in Section 4.

3.3 Regression (in a Nutshell)

Our method for the regression is based on the following observation: given a binary representation x and a threshold τ, the cardinality to be estimated can be divided into $(\tau+1)$ parts, each representing the cardinality for a distance i, $i \in [0, \tau]$. This suggests that we can learn $(\tau + 1)$ estimators, each for a distance i, and then sum them up. This method has the following advantages over existing methods:

- Compared to existing deep regression models that directly learn the relationship between the cardinality and the threshold, our method exploits the *incremental property* of cardinality: when the threshold increases from i to $i + 1$, the increment of cardinality is the cardinality for the distance $i + 1$. By learning the relationship between the increment and the distance, our method delivers better accuracy and becomes robust for queries or thresholds that are not covered by the training data.
- As we will explain later, this incremental design guarantees the monotonicity of cardinality estimation, provided that all the estimators are deterministic.
- We are able to control the size of the model by setting the maximum number of estimators. Thus, working with the feature extraction, the regression achieves fast speed even if the original threshold is large.

Based on the above observation, we learn $\tau + 1$ non-negative functions $g_0(\mathbf{x}), \ldots, g_\tau(\mathbf{x})$, each $g_i(\mathbf{x})$ estimating the cardinality of the set of records whose distances to x are exactly i. Then we have

$$g(\mathbf{x}, \tau) = \sum_{i=0}^{\tau} g_i(\mathbf{x}). \tag{1}$$

We employ the encoder-decoder model for regression. First, $\mathbf{x}$ is embedded to a dense vector space, in which the correlations of vectors are easier to learn than sparse high-dimensional binary vectors. We also encode distance i and combine it with the embedding of $\mathbf{x}$. The resultant embedding, which encodes both $\mathbf{x}$ and i, is input to the decoder g_i. By carefully choosing the encoder and the decoders, we can guarantee the monoiticity of cardinality estimation. The details will be given in Section 5. Before that, we show some case studies of feature extraction.

4. CASE STUDIES FOR FEATURE EXTRACTION

In this section, we show a series of options for feature extraction as well as some case studies.

As stated in Section 3.2, for good accuracy, a desirable feature extraction is that the Hamming distance between the binary vectors can capture the semantics of the original distance function. We provide a few example options based on existing techniques for similarity search.

- **Equivalency**: Some distance constraints can be equivalently expressed in a Hamming space, e.g., L_1 distance [22].
- **LSH**: We use d hash functions in the locality sensitive hashing (LSH) family [22], each hashing a record to a bit. $\mathbf{x}$ and $\mathbf{y}$ agree on a bit with high probability if $f(x, y) \leq \theta$, thus yielding a small Hamming distance between their binary vectors.
- **Bounding**: A bound is derived to obtain a necessary condition for the original distance constraint. E.g., $f(x, y) \leq \theta \implies H(\mathbf{x}, \mathbf{y}) \leq \tau$, where $H(\cdot, \cdot)$ denotes the Hamming distance and τ is a threshold.

For the equivalency method, since the conversion to Hamming distance is lossless, we may use it atop of the other two. This is useful when the output of the hash function or the bound is not in a Hamming space. Note that our model is not limited to these options. Other feature extraction methods, such as embedding [62], can be also used here.

As for threshold mapping, we have two parameters: $\theta_{\max}$, the maximum threshold we are going to support, and $\tau_{\max}$, a tunable parameter to control the size of our model. Any threshold $\theta \in [0, \theta_{\max}]$ is monotonically mapped to an integer $\tau \in [0, \tau_{\max}]$. In our case studies, we consider using a mapping proportional to the (expected/bounded) Hamming distance between binary representations. Note that $\theta_{\max}$ is not necessarily mapped to $\tau_{\max}$, because for integer-valued distance functions, the number of thresholds is smaller than $\tau_{\max}+1$ when $\theta_{\max} < \tau_{\max}$, meaning that only $(\theta_{\max}+1)$ decoders are useful. Next we show four case studies for some common data types and distance functions.

4.1 Hamming Distance

We first consider binary vector data and Hamming distance as the input distance function. The original data are directly fed to our regression model. Despite the possibly high dimensionality for some applications, the representation network Γ is able to handle it and convert the original binary vectors to dense representations. We will investigate this performance in the experiments.

Since the function is already Hamming distance, we use the original threshold θ as τ, if $\theta_{\max} \leq \tau_{\max}$. Otherwise, we map $\theta_{\max}$ to $\tau_{\max}$, and other thresholds are mapped proportionally; i.e., $\tau = \lfloor \tau_{\max} \cdot \theta / \theta_{\max} \rfloor$.

4.2 Levenshtein Distance

The Levenshtein distance measures the minimum number of operations, including insertion, deletion, and substitution of a character, to transform one string to another.

The feature extraction is based on bounding. The basic idea is to map each character to $(2\tau_{\max}+1)$ bits, hence to cover the effect of insertion and deletion. Let Σ denote the alphabet of strings, and $l_{\max}$ denote the maximum string length in $\mathcal{D}$. Each binary vector has $d = ((l_{\max}+2\tau_{\max}) \cdot |\Sigma|)$ bits. They are divided into $|\Sigma|$ groups, each group representing a character in Σ. Here, for simplicity, we let the subscript of a string start from 0, and the subscript of each group of the binary vector start from $-\tau_{\max}$. All the bits are initialized as 0. Given a string x, for each character σ at position i of the string, we set the j-th bit in the σ-th group to 1, where j iterates through $i-\tau_{\max}$ to $i+\tau_{\max}$. E.g., consider a string $x = \mathtt{abc}$, $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, $l_{\max} = 4$, and $\tau_{max} = 1$. The binary vector is $111000, 011100, 001110, 000000$ (groups separated by comma).

It can be proved that an edit operation makes the binary vectors differ by at most $(4\tau_{\max}+2)$ bits. Hence θ edit operations yield a Hamming distance at most $(4\tau_{\max}+2) \cdot \theta$. This upper bound is proportional to θ. Thus, we use the same threshold mapping as the one for Hamming distance.

4.3 Jaccard Distance

Given two sets x and y, the Jaccard similarity is defined as $|x \cap y| / |x \cup y|$. For ease of exposition, we use its distance form: $f(x,y) = 1 - |x \cap y| / |x \cup y|$.

We use LSH (b-bit minwise hashing [41]) for feature extraction. Given a record x, $\pi(x)$ orders the elements of x by a permutation π on the record universe $\mathcal{O}$. We use a set of k uniformly chosen permutations $\{\pi_1, \ldots, \pi_k\}$. Let $bmin(\pi(x))$ denote the last b bits of the smallest element of $\pi(x)$. We regard $bmin(\pi(x))$ as an integer in $[0, 2^b-1]$ and transform it to a Hamming space. Let $set\_bit(i,j)$ produce a one-hot i-dimensional binary vector such that only the j-th bit is 1. x is converted to a d-dimensional ($d = 2^b k$) binary vector: $[set\_bit(bmin(\pi_1(x)), 2^b); \ldots; set\_bit(bmin(\pi_k(x)), 2^b)]$. E.g., consider a set $x = \{1,2,4\}$. $\mathcal{O} = \{1,2,3,4,5\}$. $\pi_1 = 12345$, $\pi_2 = 54321$, and $\pi_3 = 21453$. $b = 2$. We have $bmin(\pi_1(x)) = 1$, $bmin(\pi_2(x)) = 0$, and $bmin(\pi_3(x)) = 2$. Suppose the set_bit operation counts from the lowest bit, starting from 0. The binary vector is $0010, 0001, 0100$ (hash functions separated by comma).

Given two sets x and y, the probability that $bmin(\pi(x)) = bmin(\pi(y))$ equals to $1 - f(x,y)$ [41]. The expected Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ is thus $f(x,y) \cdot d$. Since it is proportional to $f(x,y)$, we use the following threshold mapping: $\tau = \lfloor \tau_{\max} \cdot \theta / \theta_{\max} \rfloor$.

4.4 Euclidean Distance

We use LSH (based on p-stable distribution [20]) to handle Euclidean distance on real-valued vectors. The hash function is $h_{\mathbf{a},b}(x) = \lfloor \frac{\mathbf{a}x+b}{r} \rfloor$, where $\mathbf{a}$ is a $|x|$-dimensional vector with each element independently drawn by a normal distribution $\mathcal{N}(0,1)$, b is a real number chosen uniformly from $[0,r]$, and r is a predefined constant value. Let v denote the maximum possible hash value. We use the aforementioned set_bit function to transform the hash values to a Hamming space. Given k hash functions, x is converted to a d-dimensional ($d = k(v+1)$) binary vector: $[set\_bit(h_{\mathbf{a}_1,b_1}(x), v+1); \ldots; set\_bit(h_{\mathbf{a}_k,b_k}(x), v+1)]$. E.g., consider a record $x = [0.1, 0.2, 0.4]$, and $v = 4$. $h_{\mathbf{a}_1,b_1}(x) = 1$, $h_{\mathbf{a}_2,b_2}(x) = 3$, $h_{\mathbf{a}_3,b_3}(x) = 4$. Suppose the set_bit operation counts from the lowest bit, starting from 0. The binary vector is $00010, 01000, 10000$ (hash functions separated by comma).

Given two records x and y such that $f(x,y) = \theta$, the probability that two hash values match is

$$\begin{aligned} &Pr\{h_{\mathbf{a},b}(x) = h_{\mathbf{a},b}(y)\} = \epsilon(\theta) \\ &\quad = 1 - 2 \cdot norm(-r/\theta) - \frac{2}{\sqrt{2\pi} r/\theta}(1 - e^{-(r^2/2\theta^2)}), \end{aligned}$$

where $norm(\cdot)$ is the cumulative distribution function for a random variable with normal distribution $\mathcal{N}(0,1)$ [20]. Hence the expected Hamming distance between their binary representations is $(1-\epsilon(\theta)) \cdot d$. The threshold mapping is $\tau = \lfloor \tau_{\max} \cdot \frac{1-\epsilon(\theta)}{1-\epsilon(\theta_{\max})} \rfloor$.

5. REGRESSION MODEL IN DETAIL

We present the detailed regression model design in this section. Figure 1 shows the framework of our encoder-decoder-based regression model. $\mathbf{x}$ and τ are input to the neural network Ψ, which returns $\tau+1$ embeddings $\mathbf{z}_x^i$. Then each of the $\tau+1$ decoders takes an embeding as input and returns the cardinality in distance range $[i, i+1]$, $i \in [0, \tau-1]$. Then the $\tau+1$ cardinalities are summed up to get the final result. For easy of exposition, we first introduce the decoders g_i and then the encoder Ψ.

5.1 Decoder

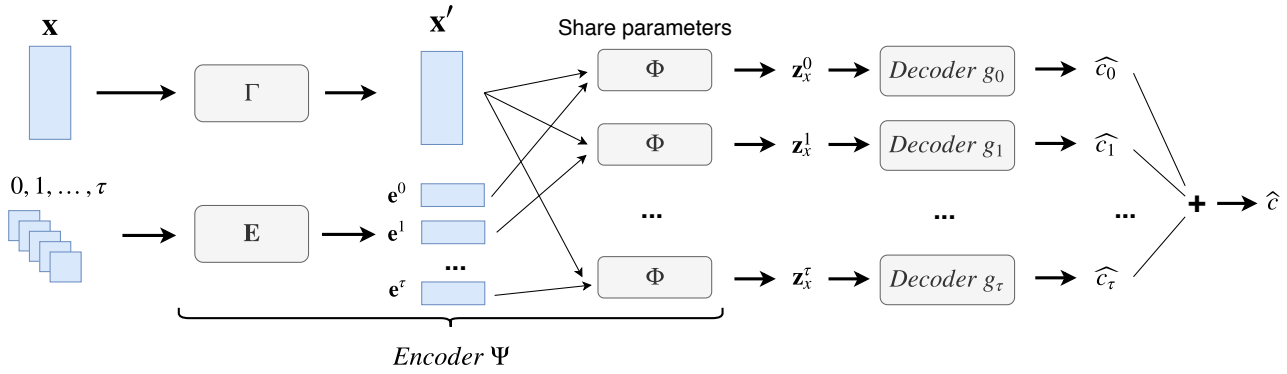


Figure 1: The Regression Model

We first assume that the encoder Ψ converts the binary vector $\mathbf{x}$ to a dense real-valued vector (called the embedding of $\mathbf{x}$). Then $g_i(\mathbf{x})$ is modeled using the ReLU activation function.

$$g_i(\mathbf{x}) = \mathsf{ReLU}(\mathbf{w}_i^{\mathrm{T}}\Psi(\mathbf{x}) + b_i), \tag{2}$$

where $\mathbf{w}_i$ and b_i are parameters of the mapping from the embedding of $\mathbf{x}$ to the cardinality estimation of distance i. The input of the ReLU function is a linear transformation of the embedding of $\mathbf{x}$, because from the machine learning perspective, if a representation of input features is well learned, then a linear model is capable of making final decisions [14]. By Equation 1, the output of the τ decoders are summed up to obtain the estimated cardinality. We have the following lemma.

LEMMA 2. *Given a deterministic function $\Psi(\cdot)$, $g(\mathbf{x}, \tau)$ is monotonically increasing with τ.*

PROOF. Consider $\langle \mathbf{x}, \tau_1 \rangle$ and $\langle \mathbf{x}, \tau_2 \rangle$, and $\tau_1 \leq \tau_2$. $g(\mathbf{x}, \tau_2) - g(\mathbf{x}, \tau_1) = \sum_{i=\tau_1+1}^{\tau_2} \mathsf{ReLU}(\mathbf{w}_i^{\mathrm{T}}\Psi(\mathbf{x}) + b_i)$. Because $\mathsf{ReLU}(\cdot) \geq 0$ and $\Psi(\cdot)$ is deterministic w.r.t. $\mathbf{x}$, $g(\mathbf{x}, \tau_2) - g(\mathbf{x}, \tau_1) \geq 0$. Thus, $g(\mathbf{x}, \tau)$ is monotonically increasing with τ. □

The lemma states that this model satisfies the requirement in Lemma 1, hence leading to the overall monotonicity.

5.2 Encoder

Recall that we assume the encoder Ψ only takes $\mathbf{x}$ as input. A major drawback of such design is that if the cardinalities of two records x_1 and x_2 are close for the distance values in a range $[\tau_1, \tau_2]$ covered by the training examples, their embeddings are likely to become similar, because the trained encoder may mistakenly regard x_1 and x_2 as similar. This may cause $g_i(\mathbf{x}_1) \approx g_i(\mathbf{x}_2)$ for $i \notin [\tau_1, \tau_2]$, i.e., the distance values not covered by the training examples, even if their actual cardinalities significantly differ. To remedy this, we design a neural network $\Psi(\cdot, \cdot)$ that takes both $\mathbf{x}$ and distance i as input [1]; i.e., embedding $\mathbf{z}_x^i = \Psi(\mathbf{x}, i)$. Then for $\mathbf{x}$ and τ, $\tau+1$ embeddings are generated: $\mathbf{z}_x^0, \mathbf{z}_x^1, \ldots, \mathbf{z}_x^\tau$. Then

$$g_i(\mathbf{x}) = \mathsf{ReLU}(\mathbf{w}_i^{\mathrm{T}}\Psi(\mathbf{x}, i) + b_i). \tag{3}$$

This still guarantees the monotonicity:

LEMMA 3. *Given a deterministic function $\Psi(\cdot, \cdot)$, $g(\mathbf{x}, \tau)$ is monotonically increasing with τ.*

[1]Instead of training $\tau + 1$ encoders, we use one encoder by sharing parameters to prevent overfitting.

The proof is very similar to Lemma 2 and omitted here.

To encode both $\mathbf{x}$ and distance i to embedding $\mathbf{z}_x^i$, Ψ includes a representation network Γ that maps $\mathbf{x}$ to a dense vector space, a distance embedding layer $\mathbf{E}$, and a shared neural network Φ that outputs the embedding $\mathbf{z}_x^i$. Next we introduce the details of these components.

5.2.1 Representation Network

Given a binary representation $\mathbf{x}$ generated by feature extraction function $h(\cdot, \cdot)$, we design a neural network Γ that maps $\mathbf{x}$ to another vector space: $\mathbf{x}' = \Gamma(\mathbf{x})$, because the correlations of sparse high-dimensional binary vectors are difficult to learn. Variational auto-encoder (VAE) [33] is a generative model to estimate data distribution by unsupervised learning. We use the latent layer of VAE, which serves to generalize the distribution of original vector space, to produce a dense representation, denoted by $VAE(\mathbf{x}, \epsilon)$. Here, ϵ is a random noise generated by normal distribution $\mathcal{N}(0, \mathbf{I})$. Γ concatenates $\mathbf{x}$ and the output of VAE; i.e., $\mathbf{x}' = [\mathbf{x}; VAE(\mathbf{x}, \epsilon)]$.

Due to the noise ϵ, the output of VAE becomes nondeterministic. Since we need a deterministic output to guarantee the monotonicity, we make a deterministic inference [54] (i.e., cardinality estimation for online queries): $\mathbf{x}' = [\mathbf{x}; \mathrm{E}_{\epsilon \sim \mathcal{N}(0,\mathbf{I})}[VAE(\mathbf{x}, \epsilon)]]$, where $\mathrm{E}[\cdot]$ denotes the expected value. Note that we still use the nondeterminstic $\mathbf{x}' = [\mathbf{x}; VAE(\mathbf{x}, \epsilon)]$ for training, because the noise can make our model generalize to unseen records and thresholds.

5.2.2 Distance Embedding

In order to embed $\mathbf{x}$ and distance i into the same vector, we design a distance embedding layer (a matrix) $\mathbf{E}$ to embed each distance i. Each column in $\mathbf{E}$ represents a distance embedding, i.e., $\mathbf{e}^i = \mathbf{E}[*, i]$. $\mathbf{E}$ is initialized randomly, following standard normal distribution. It is updated during the training procedure.

The distance embedding $\mathbf{e}^i$ is concatenated with $\mathbf{x}'$: $\mathbf{x}^i = [\mathbf{x}'; \mathbf{e}^i]$. Then we use a feedforward neural network Φ to generate embeddings $\mathbf{z}_x^i = \Phi(\mathbf{x}^i)$.

6. MODEL TRAINING

6.1 Generating Data

We uniformly sample data from dataset $\mathcal{D}$, and randomly choose training, validation, and testing sets. Then we randomly generate a set of thresholds in $[0, \theta_{\max}]$, denoted by S. For each query record x in the training set, we iterate

through all the thresholds θ in S and compute the cardinality c with a similarity selection algorithm. Then $\langle x, \theta, c \rangle$ is used a training example. We randomly choose thresholds in S for validation and from $[0, \theta_{\max}]$ for testing.

6.2 Loss Function & Dynamic Training

The loss function is defined as follows.

$$\mathcal{L}(\widehat{\mathbf{c}}, \mathbf{c}) = \mathrm{E}_{\tau \sim P(\cdot)}[\mathcal{L}_{nn}(\widehat{\mathbf{c}}, \mathbf{c})] + \lambda \mathcal{L}_{vae}(\mathbf{x}),$$

where $\mathcal{L}_{nn}(\cdot, \cdot)$ is the loss of regression model, and $\mathcal{L}_{vae}(\cdot)$ is the loss of VAE [33]. $\widehat{\mathbf{c}}$ and $\mathbf{c}$ are two vectors, each dimension representing the estimated and the real cardinalities of a set of training examples. λ is a positive hyperparameter to control the importance of VAE. To consider the interest of users in different thresholds, we introduce the probability $P(\tau)$ to indicate the importance of threshold, i.e., the probability of τ (mapped from one or more θ) selected by users in similarity selection. $P(\tau)$ is approximated using the empirical probability of thresholds in the validation set, i.e., $P(\tau) \approx \frac{\sum_{i \in \mathcal{T}^{valid}} 1\{ i=\tau \}}{|\mathcal{T}^{valid}|}$, where $\mathcal{T}^{valid}$ denotes the validation set, and $1\{ \}$ is the indicator function.

For $\mathcal{L}_{nn}$, instead of using MSE or MAPE, we resort to the mean squared logarithmic error (MSLE) with the following reason: MSLE is an approximation of MAPE [49] but MSLE narrows down the large output space to a smaller one, hence decreasing the learning difficulty.

Then we propose a dynamic training strategy for better accuracy. Given a set of training examples, let $\mathbf{c}_0, \ldots, \mathbf{c}_\tau$ and $\widehat{\mathbf{c}_0}, \ldots, \widehat{\mathbf{c}_\tau}$ denote the cardinalities and the estimated values for distance 0, ..., τ in these training examples, respectively. Simply using MSLE has the following drawback: the gradients of $\widehat{\mathbf{c}_0}$, $\cdot$, $\widehat{\mathbf{c}_\tau}$ are treated equally through the backpropagation. However, some of them may be much worse estimations than others and compromise the overall performance. The training procedure should gradually focus on training these bad estimations.

To address the above issue, besides the MSLE, we also consider the loss caused by the estimation for distance i:

$$\mathcal{L}_{nn}(\widehat{\mathbf{c}}, \mathbf{c}) = MSLE(\widehat{\mathbf{c}}, \mathbf{c}) + \lambda_\Delta \cdot \left(\sum_{i=0}^{\tau_{max}} \omega_i \cdot MSLE(\widehat{\mathbf{c}_i}, \mathbf{c}_i) \right), \tag{4}$$

where λ_Δ, ω_i are hyperparameters, and $\sum_{i=0}^{\tau_{max}} \omega_i = 1$. ω_i controls the importance of each estimation for distance i. λ_Δ controls the impact of the losses of all the estimations for $i \in [0, \tau_{\max}]$.

Due to the nonconvexity of $\mathcal{L}_{nn}$, it is difficult to find the correct direction of gradient that reaches the global or a good local optimum. Nonetheless, we can adjust its direction by considering the loss trend of the estimation for distance i, hence to encourage the model to generalize rather than to overfit the training data. Let $\ell_i(t) = MSLE(\widehat{\mathbf{c}_i}, \mathbf{c}_i)$ denote the loss of the estimation for distance i in the t-th iteration of validation. We calculate the loss trend $\Delta\ell_i(t) = \ell_i(t) - \ell_i(t-1)$. Then we design two rules for ω_i.

- If $\Delta\ell_i(t) \leq 0$, $\omega_i = 0$. That is, if the loss decreases from iteration $t-1$ to t, its current gradient has already been towards the correct direction with high probability, and thus we do not add additional gradients.

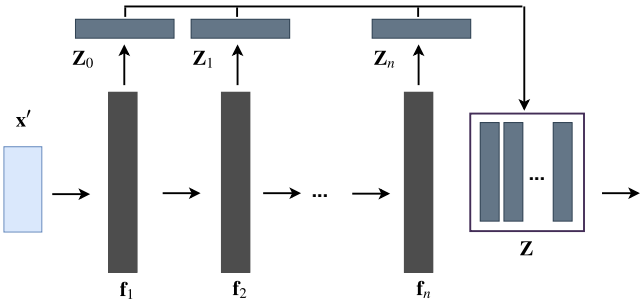


Figure 2: Φ' in Accelerated Regression Model

- If $\Delta\ell_i(t) > 0$, $\omega_i = \frac{\Delta\ell_i(t)}{\sum_A \Delta\ell_i(t),\, A=\{\, i | \Delta\ell_i(t)>0, 0 \leq i \leq \tau_{\max} \,\}}$. That is, if the loss increases, the current model we add more gradients to where the loss occurs.

In the training procedure, we first train VAE for some epochs. Then, we train the regression model with the basic loss function by MSLE for several epochs. Finally, we use the loss function in Equation 4 and calculate the hyperparameters ω_i using the validation set.

7. ACCELERATION FOR ESTIMATION

Recall in the regression model, we pair $\mathbf{x}'$ and $(\tau+1)$ distance embeddings in encoder Ψ to produce embedings $\mathbf{z}_x^i$. This leads to high computation cost for online cardinality estimation when τ is large. To reduce the cost, we propose an accelerated model, using a neural network Φ' to replace Φ and the distance embedding layer $\mathbf{E}$ to output the $(\tau_{\max}+1)$ $\mathbf{z}_x^i$ embeddings together [2]. Φ' only takes an input $\mathbf{x}'$ and reduces the computation cost from $O((\tau_{max}+1)|\Phi|)$ to $O(|\Phi'|)$.

In Figure 2, we show the framework of Φ'. It is a feedforward neural network that consists of n hidden layers $\mathbf{f}_1$, $\mathbf{f}_2$, ..., $\mathbf{f}_n$, each outputting some dimensions of the embeddings $\mathbf{z}_x^i$. In particular, each embedding $\mathbf{z}_x^i$ is partitioned into n regions, denoted by $\mathbf{z}_x^i[r_0 \,..\, r_1]$, $\mathbf{z}_x^i[r_1 \,..\, r_2]$, ..., $\mathbf{z}_x^i[r_{n-1} \,..\, r_n]$, where $0 = r_0 \leq r_1 \leq \ldots \leq r_n$ and r_n equals to the dimensionality of $\mathbf{z}_x^i$. A hidden layer $\mathbf{f}_j$ outputs the j-th region of $\mathbf{z}_x^i$, $i \in [0, \tau_{\max}]$; i.e.,

$$\mathbf{Z}_j = [\mathbf{z}_x^0[r_{j-1} \,..\, r_j] : \mathbf{z}_x^1[r_{j-1} \,..\, r_j] : \ldots : \mathbf{z}_x^{\tau_{max}}[r_{j-1} \,..\, r_j]].$$

Then we concatenate all the regions: $\mathbf{Z} = [\mathbf{Z}_1 : \mathbf{Z}_2 : \ldots : \mathbf{Z}_n]$. Each row of $\mathbf{Z}$ is an embbedding fed into the decoder; i.e., $\mathbf{z}_x^i = \mathbf{Z}[i, *]$.

In addition to fast estimation, the advantages of Φ' are as follows. First, the parameters of each hidden layer $\mathbf{f}_j$ are updated from its following layer $\mathbf{f}_{j+1}$ and the final embedding matrix $\mathbf{Z}$ through backpropagation, hence increasing the ability to learn good embeddings. Second, since each embedding is affected by all the hidden layers in Φ', the model is more likely to reach a better local optimum during the training procedure. Third, in contrast to only one output layer, all the hidden layers output embeddings, so that gradient vanishing and overfitting can be prevented.

[2]We output $(\tau_{\max}+1)$ embeddings because it is a fixed value for all queries, hence to favor implementation. Only the first $(\tau+1)$ embedding are used for threshold τ.

8. EXPERIMENTS

We report experimental results and analyses.

8.1 Setting Up

We use ten datasets for four distance functions: Hamming distance (HM), Levenshtein (edit) distance (ED), Jaccard distance (JC), and Euclidean distance (EU).

The statistics of the datasets is shown in Table 2. `Source` denotes the name of the original dataset. `Process` indicates how we process the dataset. E.g., HM-ImageNet is from the ImageNet dataset [51], and then HashNet [15], a deep hash model, is adopted to convert each image to a hash code. We uniformly sample 10% data from dataset $\mathcal{D}$, and randomly choose 80% records from the sample for training, 10% for validation, and 10% for testing.

Our models are referred to as CardNet and CardNet-L. The latter is equipped with the acceleration technique propose in Section 7. We compare with four categories of methods. (1) Traditional machine learning methods: ML-XGB [17] and ML-KDE [46]. (2) Database methods: DB-Histogram [50] for Hamming distance; DB-LSH [60] for Euclidean distance (i.e., Cosine distance in the case of normalized data); DB-Sepia [30] for Levenshtein distance,;and DB-LC [38] for Jaccard distance. Another method (referred to as DB-US) is to uniformly sample 1% records from the original data, and calculate the cardinality of results according to the samples. (3) Deep monotonic models: Mono-DLN [61]. (4) Deep regression models: DL-DNN, a feedforward neural network that has four hidden layers; DL-DNNs$_\tau$, which independently learns $(\tau_{max}+1)$ deep neural networks, each in charge of a threshold range (computed using the threshold mapping function of our model) for cardinality estimation; DL-MoE [52]; and DL-RMI [35]. Machine learning models (except ML-KDE) have the same inputs as our models for Hamming, Levenshtein, and Jaccard distance. For Euclidean distance, they are fed with the original real vectors. To compare the performance with sequence models for Levenshtein distance, we replace the neural network Γ in our models with a character-level bidirectional LSTM model [18] (referred to as BiLSTM-CardNet and BiLSTM-CardNet-L).

Parameter settings. The VAE consists of a three-layer encoder and a three-layer decoder. We use a four-layer neural network for Φ. For other values of hyperparameters (e.g., the dimensionality of $\mathbf{z}_x^i$), we fine-tune the entire network using the training and validation dataset. To train CardNet and CardNet-L, we use stochastic gradient descent and set the initial learning rate as 0.001. We reset it as 0.00025 at epoch 500, and then decrease its value per epoch. The weight decay of learning rate is $\gamma = 5 \cdot 10^{-4}$. The network is trained for 800 epochs and the prediction with the smallest validation error is selected for the evaluation with testing data. As for feature extraction, we use 256 hash functions for Jaccard distance, 256 (on EU-Glove$_{50}$) and 512 (on EU-Glove$_{300}$) hash functions for Euclidean distance. The dimensionality of the output of VAE is 40, 60, 128, 128, 128, 64, 64, 64, 64, 128, as per the order in Table 2. The dimensionality of distance embedding is 5. The dimensionality of the final embedding $\mathbf{z}_x^i$ is 60.

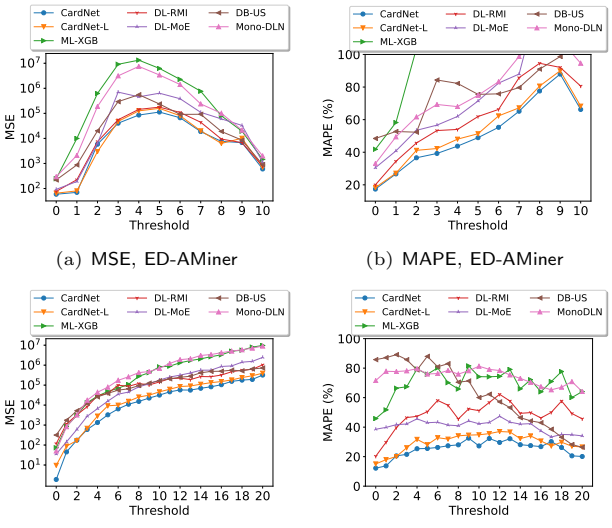
The experiments were carried out on a server with a Intel Xeon E5-2640 @2.40GHz Processor, GPU GeForce GTX 1080 Ti, and 256GB RAM running Ubuntu 16.04.4 LTS. We implemented deep models in TensorFlow, database methods in C++, ML-KDE in Java, and ML-XGB in Python using the XGBoost library [8].

8.2 Accuracy

In Tables 3 – 6, we evaluate the accuracy of our models for the four distance functions. MSE and MAPE are used as evaluation metrics. We train all the other existing models with three loss functions (MSE, MAPE and MSLE) and report the best results. The results in these tables confirm that learning incremental cardinality of distance values on binary representations of original data indeed improves the accuracy of estimation.

ML-XGB and ML-KDE do not perform very well. ML-XGB is sensitive to high dimensionality. On the 881-dimensional HM-PubChem dataset, ML-XGB has 10 times larger MSE, and 120% larger MAPE than our models. ML-KDE does not perform well in most cases. E.g., on EU-Glove$_{50}$, the MSE of ML-KDE is 52 times larger than CardNet, and the MAPE is 48% larger. This indicates that only considering the metric space is not effective for high-dimensional data.

For database methods, DB-US has small MSE in some cases, but very large MAPE. When the cardinalities are small, US achieves very bad performance. DB-Histogram avoids exponential space complexity by partitioning dimensions into sub-histograms, but the estimation is very lossy. E.g., its MSE and MAPE are 14 and 7 times larger than CardNet, respectively. For Levenshtein distance, DB-Sepia delivers at least 5 times larger MSE and 40% larger MAPE than our model. DB-LC on Jaccard distance is only capable of generating MinHash vectors with low dimensions due to its exponential time complexity. The MSE and MAPE of DB-LSH are at least 14 and 4 times larger than our model, respectively. The experimental results indicate that deep models have smaller testing errors than database methods in most cases for high-dimensional data.



(a) MSE, ED-AMiner (b) MAPE, ED-AMiner

(c) MSE, HM-RCV1 (d) MAPE, HM-RCV1

Figure 3: Testing Error v.s. Thresholds

To compare deep models, Mono-DLN has relatively the largest errors in most cases. The reason is that Mono-DLN is not efficient to generate lattice due to the only one monotonic feature. Among other deep models, DL-DNN and DL-DNNs$_\tau$ have the largest values of MSE and MAPE in most cases. This indicates that regular deep neural network

Table 2: Statistics of Datasets

Dataset	Source	Process	Data Type	Attribute	# Record	ℓ_{max}	ℓ_{avg}	Distance	θ_{max}
HM-ImageNet	ImageNet [1]	HashNet [15]	binary vector	image	1,431,167	64	64	Hamming	20
HM-RCV1	RCV1 [2]	VDSH-S [16]	binary vector	text	804,414	128	128	Hamming	20
HM-PubChem	PubChem [3]	-	binary vector	biosequence	1,000,000	881	881	Hamming	30
ED-DBLP	DBLP [4]	-	string	paper title	1,000,000	199	72.49	Levenshtein	20
ED-AMiner	AMiner [5]	-	string	author name	1,200,000	50	12.67	Levenshtein	10
JC-DBLP$_{q2}$	DBLP [4]	2-gram	set	paper title	1,000,000	198	71.49	Jaccard	0.4
JC-DBLP$_{q3}$	DBLP [4]	3-gram	set	paper title	1,000,000	197	70.49	Jaccard	0.4
JC-BMS	BMS-POS [6]	-	set	retailer set	515,597	164	6.54	Jaccard	0.4
EU-Glove$_{50}$	GloVe [7]	normalize	real-valued vector	embedding	400,000	50	50	Euclidean	0.8
EU-Glove$_{300}$	GloVe [7]	normalize	real-valued vector	embedding	1,917,494	300	300	Euclidean	0.8

Table 3: MSE and MAPE (Hamming Distance)

Metric	Models	HM-PubChem	HM-ImageNet	HM-RCV1
MAPE	ML-XGB	152.20	13.87	62.68
	ML-KDE	179.39	85.57	84.26
	DL-DNN	198.36	14.24	46.43
	DL-DNNs$_\tau$	46.43	13.00	34.53
	DL-RMI	50.57	12.36	43.72
	DL-MoE	49.47	11.93	33.76
	Mono-DLN	174.69	20.72	78.22
	DB-Histogram	79.74	56.14	61.24
	DB-US	141.04	62.51	65.03
	CardNet	**35.66**	**8.41**	**25.40**
	CardNet-L	36.57	9.63	28.68
MSE	ML-XGB	882206	12082	1637361
	ML-KDE	112952	279782	4806613
	DL-DNN	231167	10075	1540414
	DL-DNNs$_\tau$	51026	4236	1643271
	DL-RMI	42186	6774	228677
	DL-MoE	95447	7096	466048
	Mono-DLN	189743	7307	1585442
	DB-Histogram	445182	41563	5728172
	DB-US	66255	27776	246218
	CardNet	12809	**2871**	**64532**
	CardNet-L	**11598**	3044	99793

Table 4: MSE and MAPE (Edit Distance)

Metric	Models	ED-DBLP	ED-AMiner
MAPE	ML-XGB	33.26	113.68
	ML-KDE	60.23	105.17
	DL-DNN	34.12	51.36
	DL-DNNs$_\tau$	30.91	53.82
	DL-RMI	32.24	52.81
	DL-MoE	31.81	57.79
	Mono-DLN	39.10	73.48
	DB-Sepia	57.23	80.15
	DB-US	56.80	61.98
	BiLSTM-CardNet	40.95	43.44
	BiLSTM-CardNet-L	41.12	45.25
	CardNet	**22.53**	**42.26**
	CardNet-L	23.07	44.78
MSE	ML-XGB	1657	4147509
	ML-KDE	2097	3412627
	DL-DNN	1341	207286
	DL-DNNs$_\tau$	984	217193
	DL-RMI	928	93158
	DL-MoE	1235	265257
	Mono-DLN	1664	1285010
	DB-Sepia	1681	8219583
	DB-US	1095	159572
	BiLSTM-CardNet	1034	104152
	BiLSTM-CardNet-L	1061	115111
	CardNet	446	**52101**
	CardNet-L	**427**	64831

does not perform well for cardinality estimation. DL-DNNs$_\tau$ in some cases performs even worse than DL-DNN, (e.g., on EU-Glove$_{50}$, DL-DNNs$_\tau$ has 3 times larger MSE than DNN.), which indicates that sharing parameters is necessary for machine learning models and DL-DNNs$_\tau$ fails to learn the relations among thresholds. DL-RMI has the smallest values of MSE in most cases. This is because it has advantage to predict large cardinalities by hierarchically and recursively narrowing down predictions to smaller ranges. The performance of DL-RMI mainly relies on the models on upper levels. Although neural networks on upper levels discretize output space into multiple regions, they tend to mispredict the cardinalities that are closest to the boundaries between two regions. Neural networks on lower levels might receive limited number of training data, which prevents DL-RMI from generalizing well. DL-MoE has relatively small MAPE for low-dimensional data. It considers the diversity of inputs but does not directly solve the regression problem. In Levenshtein distance, BiLSTM-CardNet and BiLSTM-CardNet-L have at least 2 times larger MSE than our model in two string datasets, which shows that the bidirectional LSTM fails to well learn the property of Levenshtein distance.

CardNet and CardNet-L achieve the best performance on all the datasets, showcasing the effect of learning incremental predictions. For Hamming distance, Edit distance, Jaccard similarity, and Euclidean distance, the MSE are at least 3.6, 1.3, 5.0, and 2.1 times smaller than the best of others. Our models also reduce MAPE by at least 29.5%, 27.1%, 25.6% and 26.3%.

In Figure 3, we evaluate the accuracy by showing error distributions with varying thresholds. Here we mainly compare with the following methods: DL-MoE, Mono-DLN, DL-RMI, ML-XGB and DB-US, the better ones out of each category. Deep models achieve the best performance in most cases. CardNet and CardNet-L outperform the other models for most thresholds. They also exhibit robustness for different thresholds.

In Figure 4, we show the performance with different selectivities of cardinalities (selectivities $\leq 0.1\%$ and selectivities $> 0.1\%$). CardNet predicts large cardinality values, an important case for query optimization, better than the other models (e.g., when $\theta = 18$ to 20 in Figure 3(c) and selectivities $> 0.1\%$ in Figure 4(g)).

8.3 Efficiency of Estimation

In Table 7, we show the average estimation time of each model. CardNet-L is faster than traditional models and most deep regression models due to its small and simple model structure. Meanwhile, the time of CardNet-L is close to that of DL-DNN. E.g., on HM-PubChem, dataset both of CardNet-L and DL-DNN has 0.09ms average estimation time. This shows that the representation network and decoders bring little extra estimation time. Considering the significant accuracy improvement against DL-DNN, the small amount of extra time is acceptable. The estimation time of CardNet is very close to the other deep models and faster than the database methods and ML-KDE in most cases.

Database methods almost have the largest estimation time among all the models because they need to sample or search

Table 5: MSE and MAPE (Jaccard Distance)

Metric	Models	JC-DBLP$_{q2}$	JC-DBLP$_{q3}$	JC-BMS
MAPE	ML-XGB	6.83	6.70	34.76
	ML-KDE	38.03	37.79	42.01
	DL-DNN	13.13	5.11	43.44
	DL-DNNs$_\tau$	6.46	5.81	21.25
	DL-RMI	5.84	4.78	23.89
	DL-MoE	6.65	4.10	19.73
	Mono-DLN	5.98	6.54	49.67
	DB-LC	23.65	10.42	59.38
	DB-US	52.48	50.52	63.84
	CardNet	3.63	3.18	**11.25**
	CardNet-L	**3.53**	**3.05**	13.94
MSE	ML-XGB	21	23	3784
	ML-KDE	103	100	7236
	DL-DNN	414	138	5281
	DL-DNNs$_\tau$	721	207	7500
	DL-RMI	11	15	264
	DL-MoE	18	23	1503
	Mono-DLN	94	50	2892
	DB-LC	454	177	4725
	DB-US	45	427	6090
	CardNet	2.21	**1.98**	75
	CardNet-L	**0.59**	2.56	**64**

Table 6: MSE and MAPE (Euclidean Distance)

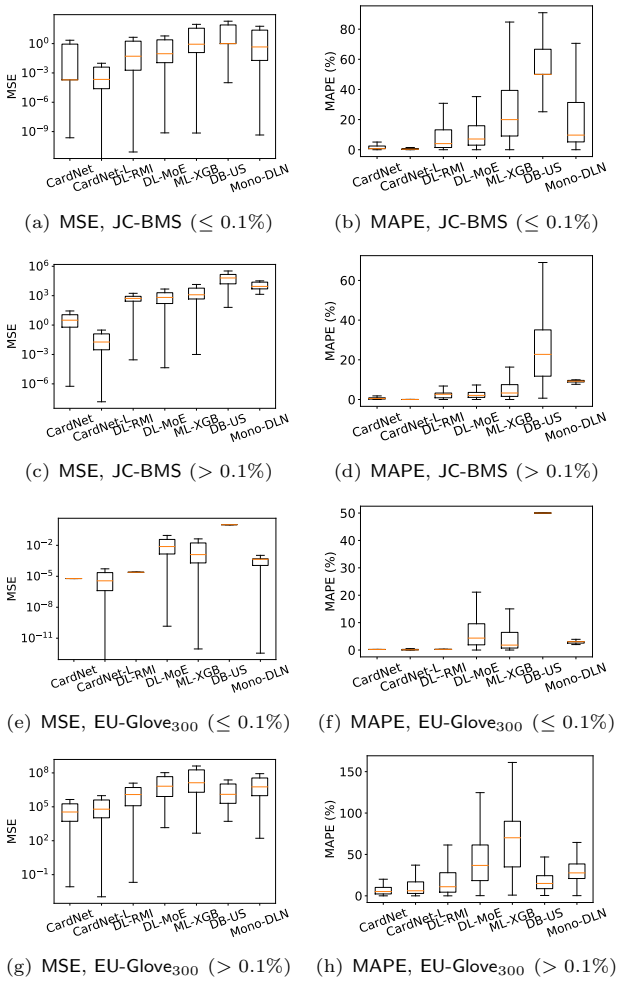
Metric	Models	EU-Glove$_{50}$	EU-Glove$_{300}$
MAPE	ML-XGB	33.87	14.46
	ML-KDE	59.84	52.38
	DL-DNN	17.57	7.24
	DL-DNNs$_\tau$	21.95	9.19
	DL-RMI	15.03	5.48
	DL-MoE	26.82	11.94
	Mono-DLN	33.95	21.67
	DB-LSH	47.12	41.91
	DB-US	98.23	112.06
	CardNet	**11.85**	**4.04**
	CardNet-L	12.71	4.58
MSE	ML-XGB	557229	821937
	ML-KDE	169604	102200
	DL-DNN	27892	1192426
	DL-DNNs$_\tau$	87991	1178239
	DL-RMI	6791	45165
	DL-MoE	315437	988918
	Mono-DLN	49389	1063687
	DB-LSH	45631	116820
	DB-US	16249	78552
	CardNet	**3245**	**6822**
	CardNet-L	3269	16809

through a prebuilt structure. E.g., DB-Histogram (DB-others) is 75 times slower than CardNet-L, because it needs to consider all combinations of thresholds among the sub-histograms, hence becoming very slow for large threshold. ML-KDE is at least 2 times slower than deep models because it needs to sample sufficient data for estimation.

We also show the performance of our models that run on GPUs (referred to as CardNet$_G$ and CardNet-L$_G$). With the speedup of GPU, both CardNet$_G$ and CardNet-L$_G$ become much faster. CardNet-L$_G$ has approximately the same estimation time with CardNet$_G$. The results show that GPUs can largely promote the speedups of deep models.

8.4 Model Size

Table 8 shows the storage sizes of the competitors. ML-KDE and DB-US have the smallest sizes, because they only need to store the sampled data and small auxiliary structures (e.g., kernels and inverted index) to assist estimation. Among the other methods, DL-DNN achieves the smallest model size in most cases. Our model sizes of are at most 3 times larger than DL-DNN, and much smaller than the other deep regression models, DL-MoE and DL-RMI. Considering the significant improvement of testing errors, the small extra storage of our models is acceptable.



(a) MSE, JC-BMS ($\leq 0.1\%$) (b) MAPE, JC-BMS ($\leq 0.1\%$)

(c) MSE, JC-BMS ($> 0.1\%$) (d) MAPE, JC-BMS ($> 0.1\%$)

(e) MSE, EU-Glove$_{300}$ ($\leq 0.1\%$) (f) MAPE, EU-Glove$_{300}$ ($\leq 0.1\%$)

(g) MSE, EU-Glove$_{300}$ ($> 0.1\%$) (h) MAPE, EU-Glove$_{300}$ ($> 0.1\%$)

Figure 4: Testing Error v.s. Cardinalities

8.5 Evaluation of Monotonicity

In Table 9, we evaluate the monotonicity by measuring DgrMon [19]: DgrMon $= \frac{\#\text{ monotonic pairs}(\mathcal{P})}{\#\text{ comparable pairs}(\mathcal{P})}$, where $\mathcal{P}$ is the set of data pairs to test. We uniformly sample 100 query records, and enumerate all the thresholds $0, 1, \ldots, \theta_{max}$ for each query record. Given a query x and two thresholds θ and $\theta + 1$, a pair is generated as $\langle \widehat{c}_\theta, \widehat{c}_{\theta+1} \rangle$, where $0 \leq \theta < \theta_{max}$, and $\widehat{c}_\theta$ is the cardinality of x with threshold θ. If a pair is monotonic, then $\widehat{c}_\theta \leq \widehat{c}_{\theta+1}$. In Table 9, ML-KDE, Mono-DLN, database methods, and our models have monotonic predictions for all pairs (i.e., 100% DgrMon). ML-KDE considers the distances between the query and the samples in metric space, so it guarantees the monotonicity. Database methods can achieve monotonicity by carefully selecting samples with different thresholds. Among all the other models, DL-RMI achieves high percentage of monotonicity. E.g., its DgrMons are larger than 90% on HM-PubChem and EU-Glove$_{50}$. This indicates its strategy of partitioning output space can preserve some monotonicity. DL-MoE does not have very good monotonicity performance,

Table 7: Average Estimation Time (microsec.)

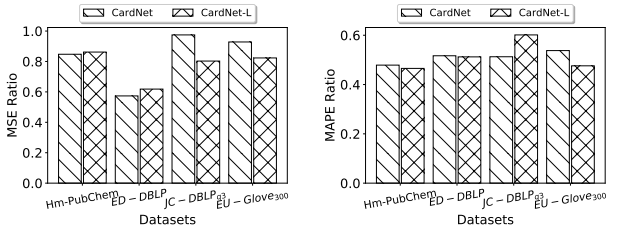
Models	HM-PubChem	HM-ImageNet	HM-RCV1	ED-DBLP	ED-AMiner	JC-DBLP$_{q2}$	JC-DBLP$_{q3}$	JC-BMS	EU-Glove$_{50}$	EU-Glove$_{300}$
ML-XGB	0.41	0.41	0.40	0.37	0.31	0.68	0.67	0.67	0.60	0.69
ML-KDE	0.96	0.83	6.30	1.24	4.73	6.80	2.35	0.97	1.22	1.28
DL-DNN	0.09	0.06	0.07	0.24	0.06	0.04	0.05	0.06	0.08	0.08
DL-DNNs$_\tau$	0.58	0.26	0.23	0.70	0.23	0.33	0.34	0.13	0.36	0.41
DL-RMI	0.46	0.37	0.28	0.54	0.36	0.41	0.42	0.37	0.53	0.68
DL-MoE	0.32	0.15	0.16	0.59	0.32	0.16	0.17	0.17	0.18	0.38
Mono-DLN	0.84	0.42	0.63	6.43	0.80	0.56	0.55	0.63	0.46	1.23
DB-method	8.50	6.20	5.98	10.01	7.64	5.01	5.78	4.67	7.34	8.45
DB-US	3.60	1.17	1.98	6.08	1.26	6.10	1.44	1.75	1.05	6.23
CardNet	0.45	0.36	0.29	0.68	0.34	0.42	0.41	0.51	0.45	0.65
CardNet-L	0.09	0.07	0.08	0.26	0.08	0.07	0.07	0.08	0.06	0.09
CardNet$_G$	0.04	0.04	0.04	0.10	0.04	0.03	0.03	0.05	0.06	0.05
CardNet-L$_G$	0.03	0.03	0.03	0.08	0.03	0.02	0.02	0.03	0.03	0.03

Table 8: Model Sizes (MB)

Models	HM-PubChem	HM-ImageNet	HM-RCV1	ED-DBLP	ED-AMiner	JC-DBLP$_{q2}$	JC-DBLP$_{q3}$	JC-BMS	EU-Glove$_{50}$	EU-Glove$_{300}$
ML-XGB	68	36	36	48	36	63	63	48	63	63
ML-KDE	8.4	4.5	7.8	7.1	1.5	7.5	7.5	3.6	7.7	18
DL-DNN	11.4	5.0	3.6	49	14.5	7	7	8	6	9.8
DL-DNNs$_\tau$	337	105	72	485	154	382	382	183	132	158
DL-RMI	80	57	67	122	84	65	65	54	60	66
DL-MoE	40	16	19	101	52	48	48	35	36	52
Mono-DLN	76	28	38	186	75	38	38	28	26	64
DB-method	143	10.4	20.8	142	31	74	74	39	86	86
DB-US	2.6	0.6	0.5	0.7	0.5	0.7	0.7	0.6	1.2	3.4
CardNet	38	9.6	12	96	40	19	19	16	21	23
CardNet-L	46	16	18	105	54	25	25	22	31	35

Table 9: Monotonicity Test (DgrMon, %)

Models	HM-PubChem	ED-DBLP	JC-BMS	EU-Glove$_{50}$
ML-XGB	59.97	45.95	74.56	81.32
ML-KDE	100.00	100.00	100.00	100.00
DL-DNN	88.52	63.43	76.78	93.61
DL-DNNs$_\tau$	69.13	58.19	48.68	68.63
DL-RMI	93.65	87.43	65.44	93.61
DL-MoE	75.71	57.72	53.05	86.42
Mono-DLN	100.00	100.00	100.00	100.00
DB-method	100.00	100.00	100.00	100.00
DB-US	100.00	100.00	100.00	100.00
CardNet	100.00	100.00	100.00	100.00
CardNet-L	100.00	100.00	100.00	100.00

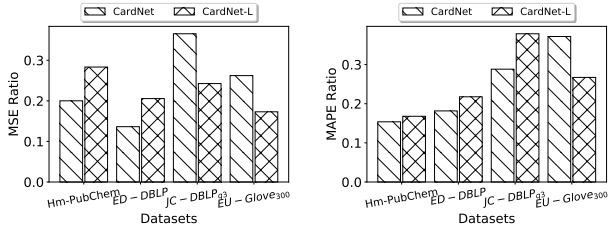


(a) MSE (b) MAPE

Figure 5: Effect of Monotonicity

because its strategy assigns inputs to proper experts but does not consider any monotonicity. Our models guarantee the monotinicity and the experiment also confirms this.

In Figure 5, we show that incremental predictions with decoders that guarantee the monotinicity can assist accuracy in four datasets. Here, we compare CardNet and CardNet-L with model DNN$^+$ that has the same structure with our models but make predictions after Φ, i.e., without decoders. To measure the improvement of incremental predictions, we design an improvement ratio: $\gamma_\xi = \frac{\xi(\text{IntCardNet-I/II}) - \xi(\text{DNN}^+)}{\xi(\text{DNN}^+)}$, where $\xi = \{\text{MSE}, \text{MAPE}\}$. The higher γ_ξ indicates the more



(a) MSE (b) MAPE

Figure 6: Effect of Dynamic Training

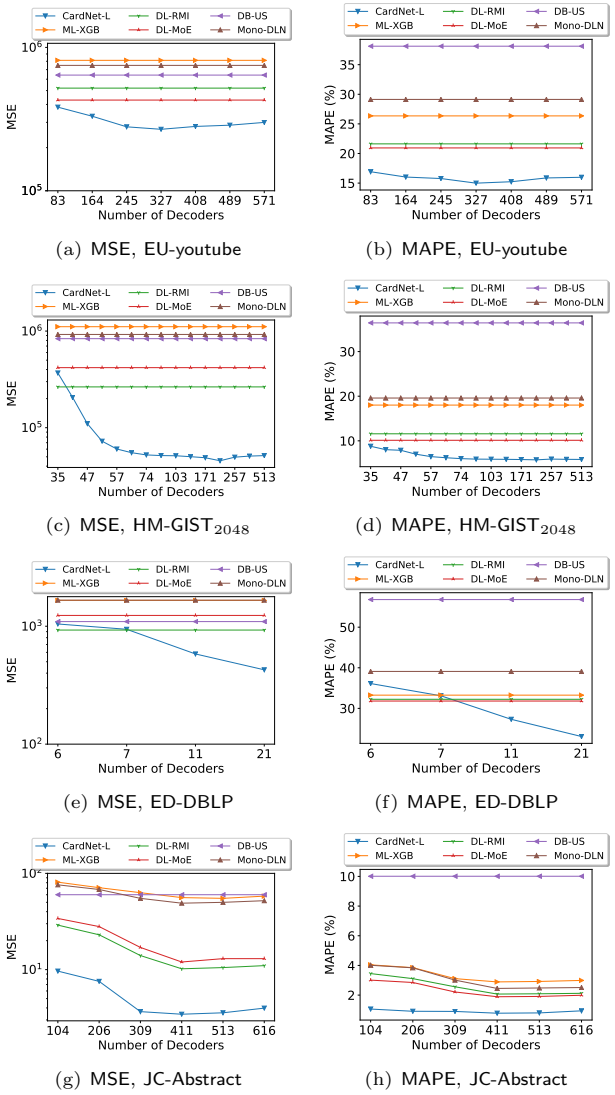
importance of decoders. Compared with DNN$^+$, our models exhibit significant improvement, especially for MSE. In Figure 5, the γ_{MSE} values on the four datasets are larger than 0.5. E.g., on JC-DBLP$_{q3}$, the γ_{MSE} of CardNet is 0.96. Our model also delivers significant improvement in terms of γ_{MAPE}. E.g., the γ_{MAPE}s of CardNet and CardNet-L on ED-DBLP are larger than 0.5. The experimental results indicate that learning incremental predictions (i.e., with decoders that guarantee the monotonicity) indeed improves the accuracy of estimation.

8.6 Effect of Dynamic Training

In Figure 6, we show the improvement of dynamic training. We still use γ_ξ as measurement and DNN$^+$ is replaced by models that have the same structure with CardNet and CardNet-L without dynamic training, i.e., $\mathcal{L}_{nn}(\cdot) = \ell_{nn}(\cdot)$. The experimental results show that dynamic training can promote the estimation of our model. E.g., on JC-DBLP$_{q3}$, the dynamic training of CardNet improves 36% over the one without dynamic training. This is because dynamic training can focus on incremental predictions of distance values that are not well learned during the training procedure, so as to achieve good performance for overall predictions.

Table 10: Statistics of Datasets with High Dimensionality

Dataset	Source	Process	Data Type	Attribute	# Record	ℓ_{max}	ℓ_{avg}	Distance	θ_{max}
EU-youtube	Youtube_Faces [9]	normalize	real-valued vector	video	346,194	1770	1770	Euclidean	0.8
HM-GIST$_{2048}$	GIST [10]	Spectral Hashing [57]	binary vector	image	982,677	2048	2048	Hamming	512
JC-Abstract	Wikipedia [11]	3-gram	string	abstract	1,150,842	732	496.06	Jaccard	0.4



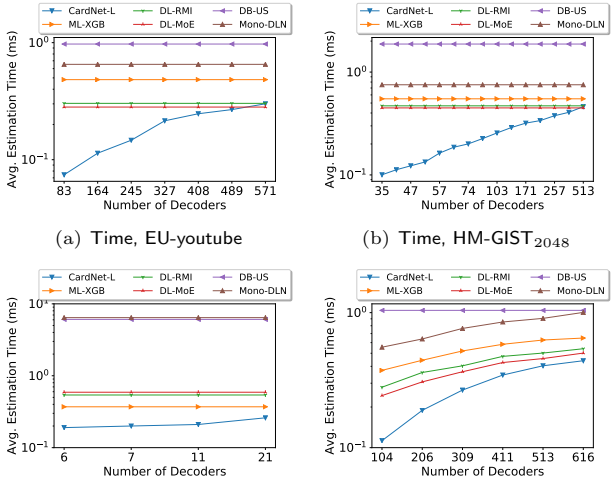
(a) MSE, EU-youtube (b) MAPE, EU-youtube

(c) MSE, HM-GIST$_{2048}$ (d) MAPE, HM-GIST$_{2048}$

(e) MSE, ED-DBLP (f) MAPE, ED-DBLP

(g) MSE, JC-Abstract (h) MAPE, JC-Abstract

Figure 7: Testing Error v.s. Number of Decoders



(a) Time, EU-youtube (b) Time, HM-GIST$_{2048}$

(c) Time, ED-DBLP (d) Time, JC-Abstract

Figure 8: Estimation Time v.s. Number of Decoders

Table 11: Testing Error v.s. Training Examples

Metric	Dataset	20%	40%	60%	80%	100%
MAPE	HM-ImageNet	11.65	10.98	10.40	9.71	9.63
	ED-AMiner	53.11	50.22	47.83	46.14	44.78
	JC-BMS	20.32	17.81	12.25	13.19	13.94
	EU-Glove$_{300}$	6.99	5.62	4.16	4.62	4.58
MSE	HM-ImageNet	4373	3486	3181	3096	3044
	ED-AMiner	91429	76647	73438	69130	64831
	JC-BMS	107	89	87	71	64
	EU-Glove$_{300}$	20770	18115	9109	15848	16809

8.7 Varying Number of Training Examples

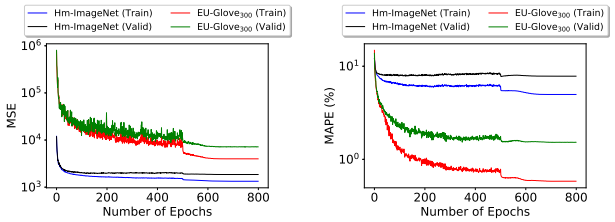
In Table 11, we show the performance of CardNet-L by varying the number of training examples. We use one dataset for each distance function, and randomly sample 20%, 40%, 60%, and 80% training data to separately train our model. The experimental results show that our model is robust with the training data size and has enough capability to generalize well. E.g., on HM-ImageNet, the MSE and MAPE of our model only increase to 1.4 and 1.2 times, respectively, when the number of training data are reduced by 80%.

8.8 Varying Epochs

In Figure 9, we show the training and validation errors of CardNet-L by varying epoch on HM-ImageNet and EU-Glove$_{300}$ datasets. At the first several epochs, CardNet-L converges quickly. At epoch 500, the learning rate is decreasing, which leads to further deceasing training and validation errors. After epoch 700, CardNet-L tends to be stable.

8.9 Varying Number of Decoders

In Figure 7, we evaluate the accuracy by varying the number of decoders. In order to show the trend clearly, besides ED-DBLP, we use three datasets with very large l_{avg} and l_{max}, whose statistics is given in Table 10. As seen from the experimental results, we discover that using the largest $\tau_{\max}$ setting does not always lead to the best performance. E.g., on EU-youtube, the best performance is achieved when $\tau_{max} = 326$ (327 decoders). When there are too few decoders, the feature extraction becomes lossy and cannot successfully capture the semantic information of the original distance functions. As the number of decoders increases, the feature extraction becomes more effective to capture the semantics. On the other hand, the performance drops if we use an excessive number of decoders. This is because given a query, the cardinality only increases at a few thresholds (e.g., a threshold of 50 and 51 might produce the same cardinality). Using too many decoders will involve too many non-increasing points, posing difficulty in learning the regression model. In Figure 8, we show the average estimation time of different models with varing number of decoders. The curves



(a) MSE (b) MAPE

Figure 9: Training and Validation Errors v.s. Epochs

of CardNet-L are approximately linear, suggesting that more decoders lead to larger estimation cost. For most settings, CardNet-L has the smallest estimation cost.

9. CONCLUSIONS

In this paper, we studied the problem of cardinality estimation of similarity selection. Observing the major drawbacks of existing methods, we designed a deep learning-based method which guarantees the monotonicity of cardinality estimation. Our method consists of two components: feature extraction and regression. The feature extraction component maps original data and threshold to Hamming space, hence to support any data types and distance functions. The regression component estimates the cardinality based on an encoder-decoder model. We exploited the incremental property of cardinality and devised a set of encoder and decoders that incrementally estimates the cardinality for each distance value in the Hamming space. We developed a training strategy specific to our model, and proposed optimization techniques to speed up online estimation. Experiments were conducted on four distance functions on ten real datasets and demonstrated the superiority of the proposed method in terms of accuracy and efficiency.

10. REFERENCES

[1] `http://www.image-net.org/`.

[2] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html`.

[3] `https://pubchem.ncbi.nlm.nih.gov/`.

[4] `https://dblp2.uni-trier.de/`.

[5] `https://aminer.org/`.

[6] `https://www.kdd.org/kdd-cup/view/kdd-cup-2000`.

[7] `https://nlp.stanford.edu/projects/glove/`.

[8] `https://xgboost.readthedocs.io/en/latest/`.

[9] `http://www.cs.tau.ac.il/~wolf/ytfaces/index.html`.

[10] `http://horatio.cs.nyu.edu/mit/tiny/data/index.html`.

[11] `https://wiki.dbpedia.org/services-resources/documentation/datasets`.

[12] I. Absalyamov, M. J. Carey, and V. J. Tsotras. Lightweight cardinality estimation in lsm-based systems. In *SIGMOD*, pages 841–855, 2018.

[13] C. Anagnostopoulos and P. Triantafillou. Query-driven learning for predictive analytics of data subspace cardinality. *ACM Transactions on Knowledge Discovery from Data*, 11(4):47, 2017.

[14] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[15] Z. Cao, M. Long, J. Wang, and S. Y. Philip. Hashnet: Deep learning to hash by continuation. In *ICCV*, pages 5609–5618, 2017.

[16] S. Chaidaroon and Y. Fang. Variational deep semantic hashing for text documents. In *SIGIR*, pages 75–84, 2017.

[17] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.

[18] J. P. Chiu and E. Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[19] H. Daniels and M. Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

[20] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SOCG*, pages 253–262, 2004.

[21] M. M. Fard, K. Canini, A. Cotter, J. Pfeifer, and M. Gupta. Fast and flexible monotonic functions with ensembles of lattices. In *NIPS*, pages 2919–2927, 2016.

[22] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[23] M. Gupta, A. Cotter, J. Pfeifer, K. Voevodski, K. Canini, A. Mangylov, W. Moczydlowski, and A. Van Esbroeck. Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research*, 17(1):3790–3836, 2016.

[24] P. J. Haas and A. N. Swami. *Sequential Sampling Procedures for Query Size Estimation*, volume 21. ACM, 1992.

[25] H. Harmouch and F. Naumann. Cardinality estimation: an experimental survey. *Proceedings of the VLDB Endowment*, 11(4):499–512, 2017.

[26] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, GPU-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, pages 1477–1492, 2015.

[27] Y. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.

[28] O. Ivanov and S. Bartunov. Adaptive cardinality estimation. *arXiv preprint arXiv:1711.08330*, 2017.

[29] H. Jiang. Uniform convergence rates for kernel density estimation. In *ICML*, pages 1694–1703, 2017.

[30] L. Jin, C. Li, and R. Vernica. Sepia: estimating selectivities of approximate string predicates in large databases. *The VLDB Journal*, 17(5):1213–1229, 2008.

[31] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, et al. Googles multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

[32] M. Kiefer, M. Heimel, S. Breß, and V. Markl. Estimating join selectivities using

bandwidth-optimized kernel density models. *Proceedings of the VLDB Endowment*, 10(13):2085–2096, Sept. 2017.

[33] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[34] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.

[35] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.

[36] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning. *arXiv preprint arXiv:1808.03196*, 2018.

[37] H. Lee, R. T. Ng, and K. Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *VLDB*, pages 195–206, 2007.

[38] H. Lee, R. T. Ng, and K. Shim. Power-law based estimation of set similarity join size. *Proceedings of the VLDB Endowment*, 2(1):658–669, 2009.

[39] H. Lee, R. T. Ng, and K. Shim. Similarity join size estimation using locality sensitive hashing. *Proceedings of the VLDB Endowment*, 4(6):338–349, 2011.

[40] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.

[41] P. Li and C. König. b-bit minwise hashing. In *WWW*, pages 671–680, 2010.

[42] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. In *PODS*, pages 40–46, 1990.

[43] H. Liu, M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte. Cardinality estimation using neural networks. In *CSSE*, pages 53–59, 2015.

[44] X. Liu, W. Liang, Y. Wang, S. Li, and M. Pei. 3d head pose estimation with convolutional neural network trained on synthetic images. In *ICIP*, pages 1289–1293, 2016.

[45] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. *arXiv preprint arXiv:1803.00055*, 2018.

[46] M. Mattig, T. Fober, C. Beilschmidt, and B. Seeger. Kernel-based cardinality estimation on metric data. In *EDBT*, pages 349–360, 2018.

[47] A. Mazeika, M. H. Böhlen, N. Koudas, and D. Srivastava. Estimating the selectivity of approximate string queries. *ACM Transactions on Database Systems*, 32(2):12, 2007.

[48] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. *arXiv preprint arXiv:1803.08604*, 2018.

[49] H. Park and L. Stefanski. Relative-error prediction. *Statistics & Probability Letters*, 40(3):227–236, 1998.

[50] J. Qin, Y. Wang, C. Xiao, W. Wang, X. Lin, and Y. Ishikawa. Gph: Similarity search in hamming space. In *ICDE*, pages 29–40, 2018.

[51] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[52] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[53] W. Shen, Y. Guo, Y. Wang, K. Zhao, B. Wang, and A. Yuille. Deep regression forests for age estimation. In *CVPR*, pages 2304–2313, 2017.

[54] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *NIPS*, pages 3483–3491, 2015.

[55] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, pages 3476–3483, 2013.

[56] I. Trummer, S. Moseley, D. Maram, S. Jo, and J. Antonakakis. Skinnerdb: regret-bounded query evaluation via reinforcement learning. *Proceedings of the VLDB Endowment*, 11(12):2074–2077, 2018.

[57] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2009.

[58] W. Wu. Sampling-based cardinality estimation algorithms: A survey and an empirical evaluation.

[59] W. Wu, J. F. Naughton, and H. Singh. Sampling-based query re-optimization. In *SIGMOD*, pages 1721–1736. ACM, 2016.

[60] X. Wu, M. Charikar, and V. Natchu. Local density estimation in high dimensions. In *ICML*, pages 5293–5301, 2018.

[61] S. You, D. Ding, K. Canini, J. Pfeifer, and M. Gupta. Deep lattice networks and partial monotonic functions. In *NIPS*, pages 2981–2989, 2017.

[62] H. Zhang and Q. Zhang. Embedjoin: Efficient edit similarity joins via embeddings. In *KDD*, pages 585–594, 2017.

[63] W. Zhang, K. Gao, Y. Zhang, and J. Li. Efficient approximate nearest neighbor search with integrated binary codes. In *ACM Multimedia*, pages 1189–1192, 2011.

[64] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539, 2018.