Consistent and Flexible Selectivity Estimation for High-Dimensional Data

ABSTRACT

Selectivity estimation aims at estimating the number of database objects that satisfy a selection criterion. Answering this problem accurately and efficiently is essential to many applications, such as density estimation, outlier detection, query optimization, and data integration. The estimation problem is especially challenging for large-scale high-dimensional data due to the curse of dimensionality, the large variance of selectivity across different queries, and the need to make the estimator consistent (i.e., the selectivity is non-decreasing in the threshold). We propose a new deep learning-based model that learns a *query-dependent piecewise linear function* as selectivity estimator, which is flexible to fit the selectivity curve of any query object and threshold, while guaranteeing that the output is non-decreasing in the threshold. To improve the accuracy for large datasets, we propose to partition the dataset into multiple disjoint subsets and build a local model on each of them. We perform experiments on real datasets and show that the proposed model significantly outperforms state-of-the-art models in accuracy and is competitive in efficiency.

1 INTRODUCTION

In this paper, we consider the following selectivity estimation problem for high-dimensional data: given a query object $\mathbf{x}$, a distance function d, and a distance threshold t, estimate the number of objects $\mathbf{o}$s in a database that satisfy $d(\mathbf{x}, \mathbf{o}) \leq t$. It is an essential procedure in estimating local density [42] and outlierness [3], which are keys to density estimation in statistics and density-based outlier detection in data mining. It is also known as the cardinality estimation problem in the database area. Accurate estimation helps to find an optimal query execution plan in databases dealing with high-dimensional data [16]; e.g., hands-off entity matching systems [8] extract paths from random forests and take each path – a conjunction of similarity predicates over multiple attributes – as a blocking rule, and efficient blocking can be achieved if we find a good query execution plan [40]. In addition, many modern recommender systems resort to latent representations (embeddings) of users and/or items [24, 47]. To make recommendation, a selection query is invoked to obtain a set of candidates to be further ranked by sophisticated models. Estimating the number of candidates helps to choose a proper ranker to improve the quality of recommendation.

Selectivity estimation for large-scale high-dimensional data is still an open problem due to the following factors: (1) *Large variance of selectivity.* The selectivity varies across queries and may differ by several orders of magnitude. A good estimator is supposed to predict accurately for both small and large selectivity values. (2) *Curse of dimensionality.* Many methods that work well on low-dimensional data, such as histograms [18], are intractable when we seek an optimal solution, and they significantly lose accuracy with the increase of dimensionality. (3) *Consistency requirement.* When the query object is fixed, selectivity is non-decreasing in the threshold. Hence users may want the estimated selectivity to be non-decreasing and interpretable in applications such as density estimation. This requirement rules out many existing methods.

To address the above challenges, we propose a novel deep regression method that guarantees consistency. We holistically approximate the selectivity curve using a *query-dependent piecewise linear function* consisting of control points that are learned from training data. This function family is *flexible* in the sense that it can closely approximate the selectivity curve of any input query object; e.g., using more control points for the part of the curve where selectivity changes rapidly. Together with a robust loss function, we are able to alleviate the impact of large variance across different queries. To handle high dimensionality, we incorporate an autoencoder that learns the latent representation of the query object with respect to the data distribution. The query object and its latent representation are fed to a query-dependent control point model, enhancing the fit to the selectivity curve of the query object. To ensure consistency, we achieve the monotonicity of estimated selectivity by converting the problem to a standard neural network prediction task, rather than imposing additional limitations such as restricting weights to be non-negative [7] or limiting to multi-linear functions [9]. To improve the accuracy on large-scale datasets, we propose a partition-based method to divide the database into multiple disjoint subsets and learn a local model on each of them. Since update may exists in the database, we employ incremental learning to cope with this issue instead of training from scratch.

We perform experiments on three real datasets. The results show that our method outperforms various state-of-the-art models. Compared to the best existing model [40], the improvement of accuracy is up to 3.7 times in mean squared error and consistent across datasets, distance functions, and error metrics. The experiments also demonstrate that our method is competitive in estimation speed and robust against update in the database.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces preliminaries. Section 4 summarizes our ideas. Section 5 presents our model. Section 6 discusses model complexity and the comparison with other models. We report experimental results in Section 7. Section 8 concludes the paper.

2 RELATED WORK

Traditional Estimation Models. Selectivity estimation has been extensively studied in database systems, where prevalent approaches are based on histograms [18], sampling [43, 45], or sketches [6]. However, few of them are applicable to high-dimensional data due to the curse of dimensionality. Wu *et al.* [44] substantially improved the sampling complexity by using locality-sensitive hashing (LSH) as a means to perform importance sampling and attain an improved unbiased estimation of selectivity. This approach is heavily tied to a handful of distance or similarity functions that have known LSH functions. Kernel density estimation (KDE) [16, 27] has been

proposed to handle selectivity estimation in metric space. Mattig *et al.* [27] alleviated the curse of dimensionality by focusing on the distribution in metric space. However, strong assumptions are usually imposed on the kernel function (e.g., only diagonal covariance matrix for Gaussian kernels), and one kernel function may be inadequate to model complex distributions in high-dimensional data.

Regression Models without Consistency Guarantee. Selectivity estimation can be formalized as an ordinary regression problem with query and threshold as input features, if the consistent requirement is not enforced. Non-deep learning-based regression models (e.g, support vector regression) do not perform well for our task due to the high dimensionality. A recent trend is to use deep learning-based regression models. Vanilla deep regression [25, 36, 37] learns good representations of input patterns. The mixture of expert model (MoE) [33] has a sparsely-gated mixture-of-experts layer that assigns data to proper experts (models) which lead to better generalization. The recursive model index (RMI) [23] is a regression model that can be used to replace the B-tree index in relational databases. Deep regression has also been used to predict selectivities (cardinalities) [21, 35] in relational databases, amid a set of recent advances in learning methods for this task [15, 29, 30, 38, 46].

Models with Consistency Guarantee. Gradient boosting trees (e.g., XGBoost [5] and LightGBM [39]) support monotonic regression. Lattice regression [9, 11, 13, 48] uses a multi-linearly interpolated lookup table to solve low-dimensional regression problems. By enforcing constraints on its parameter values, it can guarantee monotonicity. To accommodate high dimensional inputs, Fard *et al.* [9] proposed to build an ensemble of lattice using subsets of input features. To further increase the modelling power, deep lattice network (DLN) [48] was proposed to interlace non-linear calibration layers and ensemble of lattice layers. Recently, lattice regression has also been used to learn a spatial index [26]. Besides, UMNN [41] is an autoregressive flow model which adopts Clenshaw-Curtis quadrature to achieve monotonicity. Other monotonic models include isotonic regression [14, 34] and MinMaxNet [7]. CardNet [40] is a recently proposed method for monotonic selectivity estimation of similarity selection query for various data types. It maps original data to binary vectors and the threshold to an integer τ, and then predicts the selectivity for distance $[0, 1, \ldots, \tau]$ respectively with $(\tau + 1)$ encoder-decoder models. When applying to high-dimensional data, its has the following drawbacks: the mapping from the input threshold to τ is not injective, i.e., multiple thresholds may be mapped to the same τ and the same selectivity is always output for them; the overall accuracy is significantly affected if one of the $(\tau + 1)$ decoders is not accurate for some query.

3 PRELIMINARIES

Problem 1 (Selectivity Estimation for High-Dimensional Data). *Given a database of real-valued vectors $\mathcal{D} = \{ \mathbf{o}_i \}_{i=1}^{n}$, a distance function $d(\cdot, \cdot)$, a scalar threshold t, and a query object $\mathbf{x}$, estimate the selectivity in the database, i.e., $|\{ \mathbf{o} \mid d(\mathbf{x}, \mathbf{o}) \leq t, \mathbf{o} \in \mathcal{D} \}|$.*

While we assume d is a distance function, it is easy to extend it to consider d as a similarity function by changing $\leq$ to $\geq$ in the above definition. In the rest of the paper, to describe our method, we focus on the case when d is a distance function, while we evaluate Euclidean distance and cosine similarity in our experiments.

We can view the selectivity (i.e., the ground truth label) y of a query object $\mathbf{x}$ and a threshold t as generated by a function $y = f(\mathbf{x}, t, \mathcal{D})$. We call f the **value function**. Our goal is to estimate $f(\mathbf{x}, t, \mathcal{D})$ using another function $\hat{f}(\mathbf{x}, t, \mathcal{D})$.

One unique requirement of our problem is that the estimator $\hat{f}$ needs to be *consistent*: $\hat{f}$ is consistent if and only if it is *non-decreasing* in the threshold t for every query object $\mathbf{x}$; i.e., $\forall \mathbf{x}$, $\hat{f}(\mathbf{x}, t', \mathcal{D}) \geq \hat{f}(\mathbf{x}, t, \mathcal{D})$ iff. $t' \geq t$.

4 OBSERVATIONS AND IDEAS

When $|\mathcal{D}|$ is large, it is hard to estimate f directly. One of the main challenges is that f may be non-smooth with respect to the input variables. In the worst case, we have:

- For any vector $\Delta\mathbf{x}$, there exists a database $\mathcal{D}$ of n objects and a query $(\mathbf{x}, t)$ such that $f(\mathbf{x}, t, \mathcal{D}) = 0$ and $f(\mathbf{x} + \Delta\mathbf{x}, t, \mathcal{D}) = n$.
- For any $\epsilon > 0$, there exists a database $\mathcal{D}$ of n objects and a query $(\mathbf{x}, t)$ such that $f(\mathbf{x}, t, \mathcal{D}) = 0$ and $f(\mathbf{x}, t + \epsilon, \mathcal{D}) = n$.

This means any model that directly approximates f is hard.

Our idea to mitigate this issue is to reduce n: instead of estimating one function f, we estimate multiple functions such that each function's output range is a small fraction of n. More specifically, we adopt the following two partitioning schemes.

Threshold Partitioning. Assume the maximum threshold we support is $t_{\max}$. We consider dividing it with an increasing sequence of $(L+2)$ values: $[\tau_0, \tau_1, \ldots, \tau_{L+1}]$ such that $\tau_i < \tau_j$ if $i < j$, $\tau_0 = 0$, and $\tau_{L+1} = t_{\max} + \epsilon$, where ϵ is a small positive quantity [1]. Let $g_i(\mathbf{x}, t)$ be an interpolant function for interval $[\tau_{i-1}, \tau_i)$. Then we have

$$\hat{f}(\mathbf{x}, t, \mathcal{D}) = \sum_{i=1}^{L+1} \mathbf{1}[\![t \in [\tau_{i-1}, \tau_i)]\!] \cdot g_i(\mathbf{x}, t), \tag{1}$$

where $\mathbf{1}[\![]\!]$ denotes the indicator function.

Data Partitioning. We partition the database $\mathcal{D}$ into K disjoint parts $\mathcal{D}_1, \ldots, \mathcal{D}_K$, and let f_i denote the value function defined on the i-th part. Then we have $\hat{f}(\mathbf{x}, t, \mathcal{D}) = \sum_{i=1}^{K} \hat{f}_i(\mathbf{x}, t, \mathcal{D}_i)$.

In the next section, we will present a model that combines both partitioning schemes such that the partitions are adaptive to the query object and the database.

5 SELECTIVITY ESTIMATOR

5.1 Threshold Partitioning

Our idea is to approximate f using a regression model $\hat{f}(\mathbf{x}, t, \mathcal{D}; \Theta)$. Recall the sequence $[\tau_0, \tau_1, \ldots, \tau_{L+1}]$ in Section 4. We consider the family of continuous piecewise linear function to implement the interpolation $g_i(\mathbf{x}, t)$, $i \in [0, L + 1]$. A piecewise linear function is a continuous function of $(L + 1)$ pieces, each being a linear function defined on $[\tau_{i-1}, \tau_i)$. The τ_i values are called *control points*. Given a query object $\mathbf{x}$, let p_i denote the estimated selectivity for a threshold τ_i. For the g_i function in Eq. (1), we have

$$g_i(\mathbf{x}, t) = p_{i-1} + \frac{t - \tau_{i-1}}{\tau_i - \tau_{i-1}} \cdot (p_i - p_{i-1}). \tag{2}$$

[1] ϵ is used to cover the corner case of $t = t_{\max}$ in Eq. (1).

Hence the regression model is parameterized by $\Theta \stackrel{\text{def}}{=} \{(\tau_i, p_i)\}_{i=0}^{L+1}$. Note that τ_i and p_i values are dependent on $\mathbf{x}$; i.e., the piecewise linear function is query-dependent.

Using the above design for Θ has the following property to guarantee the consistency [2].

Lemma 1. *Given a database $\mathcal{D}$ and a query object $\mathbf{x}$, if $p_i \geq p_{i-1}$ for $\forall i \in [1, L+1]$, then $\hat{f}(\mathbf{x}, t, \mathcal{D}; \Theta)$ is non-decreasing in t.*

Another salient property of our model is that it is flexible in the sense that it can arbitrarily well approximate the selectivity curve. Piecewise linear functions have been well explored to fit one-dimensional curves [31]. With a sufficient number of control points, one can find an optimal piecewise linear function to fit any one-dimensional curve. The idea is that a small range of input is highly likely to be linear with the output. When $\mathbf{x}$ and $\mathcal{D}$ are fixed, the selectivity only depends on t, and thus the value function can be treated as a one-dimensional curve. To distinguish different $\mathbf{x}$, we will design a deep learning approach to learn good control points and corresponding selectivities. As such, our model not only inherits the good performance of piecewise linear function but also handles different query objects.

Estimation Loss. In the regression model, the L τ_i values and the $(L+2)$ p_i values are the parameters to be learned. We use the expected loss between f and $\hat{f}$:

$$J_{\text{est}}(\hat{f}) = \sum_{((\mathbf{x},t),y)\in\mathcal{T}_{\text{train}}} \ell(f(\mathbf{x}, t, \mathcal{D}), \hat{f}(\mathbf{x}, t, \mathcal{D})), \tag{3}$$

where $\mathcal{T}_{\text{train}}$ denotes the set of training data, and $\ell(y, \hat{y})$ is a loss function between the true selectivity y and the estimated value $\hat{y}$ of a query $(\mathbf{x}, t)$. We choose the Huber loss [17] applied to the logarithmic values of y and $\hat{y}$. To prevent numeric errors, we also pad the input by a small positive quantity ϵ. Let $r \stackrel{\text{def}}{=} \ln(y+\epsilon) - \ln(\hat{y}+\epsilon)$. Then

$$\ell(y, \hat{y}) = \begin{cases} \frac{r^2}{2} & \text{, if } |r| \leq \delta; \\ \delta(|r| - \frac{\delta}{2}) & \text{, otherwise.} \end{cases}$$

δ is set to 1.345, the standard recommended value [10]. The reason for designing such a loss function is that the selectivity may differ by several orders of magnitude for different queries. If we use the ℓ_2 loss, it encourages the model to fit large selectivities well, and if we use ℓ_1 loss, it pays more attention to small selectivities. To achieve robust prediction, we reduce the value range by logarithm and the Huber loss.

5.2 Learning Piecewise Linear Function

We choose a deep neural network to learn the piecewise linear function. It has the following advantages: (1) Deep learning is able to capture the complex patterns in control points and corresponding selectivities for accurate estimation of different queries. (2) Deep learning generalizes well on queries that are not covered by training data. (3) The training data for our problem can be unlimitedly acquired by running a selection algorithm on the dataset, and this favors deep learning which often requires large training sets.

[2] Proof is provided in Appendix A.

In our model, τ_i and p_i values are generated separately for the input query object. We also require non-negative increments between consecutive parameters to ensure they are non-decreasing. In the following, we explain the learning of τ_is and p_is, followed by the overall neural network architecture.

Control Points (τ_is). Our idea is to learn the increments between τ_is.

$$\tau_i(\mathbf{x}) = \sum_{j=0}^{i-1} \boldsymbol{\Delta}_\tau(\mathbf{x})[j], \tag{4}$$

$$\text{where } \boldsymbol{\Delta}_\tau(\mathbf{x}) = \text{Norm}_{l_2}(g^{(\tau)}(\mathbf{x})) \cdot t_{max}. \tag{5}$$

Norm_{l_2} is a normalized squared function defined as

$$\text{Norm}_{l_2}(\mathbf{t}) = [\frac{t_1^2 + \frac{\epsilon}{d}}{\mathbf{t}^{\mathrm{T}}\mathbf{t} + \epsilon}, \ldots, \frac{t_d^2 + \frac{\epsilon}{d}}{\mathbf{t}^{\mathrm{T}}\mathbf{t} + \epsilon}],$$

where ϵ is a small positive quantity to avoid dividing by zero, d is the dimensionality of $\mathbf{t}$, and t_i denotes the value of the i-th dimension of $\mathbf{t}$. The model takes $\mathbf{x}$ as input and outputs L distinct thresholds in $(0, t_{\max})$. $g^{(\tau)}$ is implemented by a neural network. Then we have a $\boldsymbol{\tau}$ vector $\boldsymbol{\tau} = [0; \tau_1; \tau_2; \ldots; \tau_L; t_{max}]$.

One may consider using $\text{Softmax}(\mathbf{t})$, which is widely used for multi-classification and (self-)attention. We choose $\text{Norm}_{l_2}(\mathbf{t})$ rather than $\text{Softmax}(\mathbf{t})$ for the following reasons: (1) Due to the exponential function in $\text{Softmax}(\mathbf{t})$, a small change of $\mathbf{t}$ might lead to large variations of the output. (2) Softmax aims to highlight the important part rather than partitioning $\mathbf{t}$, while our goal is to rationally partition the range $[0, \tau_{max}]$ into several intervals such that the piecewise linear function can fit well.

Selectivities at Control Points (p_is). We learn $(L+2)$ p_i values in a similar fashion to control points, using another neural network to implement $g^{(p)}$.

$$p_i(\mathbf{x}) = \sum_{j=0}^{i} \boldsymbol{\Delta}_p(\mathbf{x})[j], \tag{6}$$

$$\text{where } \boldsymbol{\Delta}_p(\mathbf{x}) = \text{ReLU}(g^{(p)}(\mathbf{x})). \tag{7}$$

Then we have a $\mathbf{p}$ vector $\mathbf{p} = [p_0; p_1; \ldots; p_{L+1}]$. Here, we learn $(L+1)$ increments $(p_i - p_{i-1})$ instead of directly learning $(L+2)$ p_is. Thereby, we do not have to enforce a constraint $p_{i-1} \leq p_i$ for $i \in [1, L+1]$ in the learning process, and thus the learned model can better fit the selectivity curve.

Network Architecture. We illustrate the network architecture of our model in Figure 1.

The input $\mathbf{x}$ is first transformed to $\mathbf{z}$, a latent representation obtained by an autoencoder (AE) on the entire dataset. The use of the AE encourages the model to exploit latent data and query distributions in learning the piecewise linear function, and this helps the model generalize to query objects outside the training data. To learn the latent distributions of $\mathcal{D}$, we pretrain the AE on all the objects of $\mathcal{D}$, and then continue to train the AE with the queries in the training data. Due to the use of AE, the final loss function is a linear combination of the estimation loss (Eq. (3)) and the loss of the AE for the training data (denoted by J_{AE}):

$$J(\hat{f}) = J_{\text{est}}(\hat{f}) + \lambda \cdot J_{\text{AE}}. \tag{8}$$

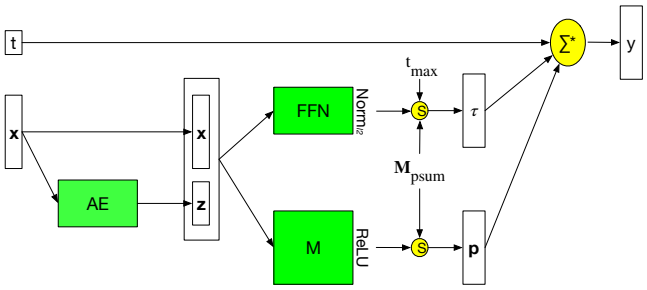


Figure 1: Network architecture.

x is concatenated with **z**, i.e., $[\mathbf{x};\mathbf{z}]$. Then $[\mathbf{x};\mathbf{z}]$ is fed into two independent neural networks: a feed-forward network (FFN) and a model M (introduced later). Two multiplications, denoted by S operators in Figure 1, are needed to separately convert the output of FFN and the output of model M to the $\boldsymbol{\tau}$ and $\mathbf{p}$ vectors, respectively. They use a scalar $t_{\max}$ and a matrix $\mathbf{M}_{\text{psum}}$ which, once multiplied on the right to a vector, perform prefix sum operation on the vector.

$$\mathbf{M}_{\text{psum}} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}.$$

The output of these networks, together with the threshold t, are fed into the operator $\sum^*$ in Figure 1, which is implemented by Eqs. (2), (5), and (7), to compute the output of the piecewise linear function, i.e., the estimated selectivity.

Model M. To achieve better performance, we learn $\mathbf{p}$ using an encoder-decoder model. In the encoder, an FFN is used to generate $(L+2)$ embeddings:

$$[\mathbf{h}_0;\mathbf{h}_1;\dots;\mathbf{h}_{L+1}] = \text{FFN}([\mathbf{x};\mathbf{z}]), \tag{9}$$

where $\mathbf{h}_i$s are high-dimensional representations. Here, we adopt $(L+2)$ embeddings, i.e., $\mathbf{h}_0,\dots,\mathbf{h}_{L+1}$, to represent the latent information of $\mathbf{p}$. In the decoder, we adopt $(L+2)$ linear transformations with the ReLU activation function:

$$k_i = \text{ReLU}(\mathbf{w}_i^{\text{T}}\mathbf{h}_i + b_i).$$

Then we have $\mathbf{p} = [k_0, k_0+k_1, \dots, \sum_{i=0}^{L+1} k_i]$.

5.3 Data Partitioning

To improve the accuracy of estimation on large-scale datasets, we divide the database into multiple disjoint subsets $\mathcal{D}_1,\dots,\mathcal{D}_K$ with approximately the same size, and build a local model on each of them. Let $\hat{f}_i$ denote each local model. Then the global model for selectivity estimation is $\hat{f} = \sum_i \hat{f}_i$.

We have considered several design choices and propose the following configuration that achieves the best empirical performance: (1) Partitioning is obtained by a cover tree-based strategy. (2) We adopt the structure in Figure 1 so that all local models share the same input representation $[\mathbf{x};\mathbf{z}]$, but each has its own neural networks to learn the control points.

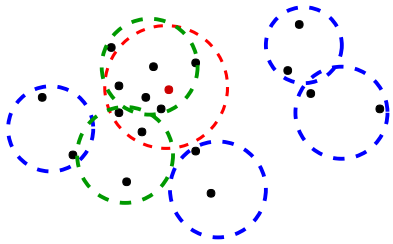


Figure 2: Data partitioning by cover tree.

Partitioning Method. We utilize a cover tree [19] to partition $\mathcal{D}$ into several parts. A partition ratio r is predefined such that the cover tree will not expand its nodes if the number of inside objects is smaller than $r|\mathcal{D}|$. Given a query $(\mathbf{x},t)$, the valid region is the circles that intersect the circle with $\mathbf{x}$ as center and t as radius. For example, in Figure 2, $\mathbf{x}$ (the red point) and t form the red circle, and data are partitioned into 6 regions. The valid region of $(\mathbf{x},t)$ is the green circles that intersect the red circle. Albeit imposing constraints, cover tree might still generate too many ball regions, i.e., leaf nodes, which lead to large number of parameters of the model and the difficulty of training. Reducing the number of ball regions is necessary. To remedy this, we adopt a merging strategy as follows. First, we still partition $\mathcal{D}$ into K' regions using cover tree. Then we cluster these regions into K ($K' \le K$) clusters $\mathcal{D}_1,\dots,\mathcal{D}_K$ by the following greedy strategy: The K' regions are sorted in decreasing order of the number of inside objects. We begin with K empty clusters. Then we scan each region and assign it to the cluster with the smallest size. The regions that belong to the same cluster are merged to one region. We consider an indicator $f_c : (\mathbf{x},t) \to \{0,1\}^K$ such that $f_c(\mathbf{x},t)[i] = 1$ if and only if the query $(\mathbf{x},t)$ intersects cluster $\mathcal{D}_i$, and employ it in our model:

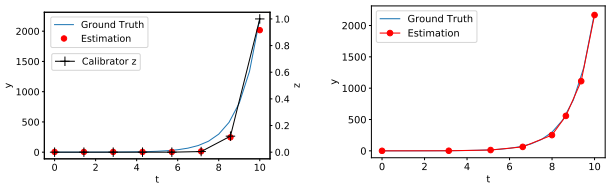
$$\hat{f}(\mathbf{x},t,\mathcal{D}) = \sum_{i=0}^{K} f_c(\mathbf{x},t)[i] \cdot \hat{f}_i(\mathbf{x},t,\mathcal{D}_i).$$

Since cover trees deal with metric spaces, for non-metric functions (e.g, cosine similarity), if possible, we equivalently convert it to a metric (e.g, Euclidean distance, as $\cos(\mathbf{u},\mathbf{v}) = 1 - \frac{\|\mathbf{u},\mathbf{v}\|^2}{2}$ for unit vectors $\mathbf{u}$ and $\mathbf{v}$). Then the cover tree partitioning still works. For those that cannot be equivalently converted to a metric, we adopt random partitioning instead of using a cover tree and modify f_c as $f_c : (\mathbf{x},t) \to \{\mathbf{1}\}^K$.

Training Procedure. We have several choices on how to train the models from multiple partitions. The default is directly training the global model $\hat{f}$, with the advantage that no extra work is needed. The other choice is to train each local model independently, using the selectivity computed on the local partition as training label. We propose yet another choice: we pretrain the local models for T epochs, and then train them jointly. In the joint training stage, we use the following loss function:

$$J_{\text{joint}} = J_{\text{est}}(\hat{f}) + \beta \cdot \sum_i J_{\text{est}}(\hat{f}_i) + \lambda \cdot J_{\text{AE}}.$$

The indicators $f_c(\cdot,\cdot)$s of all $(\mathbf{x},t)$ are precomputed before training.



(a) Simplified DLN (b) Our Model

Figure 3: Comparison of DLN and our model (both with 8 control points) to $y = f(t) = \frac{1}{10}\exp(t)$ for $t \in [0, 10]$.

5.4 Dealing with Data Updates

When the dataset $\mathcal{D}$ is updated with insertion or deletion, we first check whether our model $\hat{f}(\mathbf{x}, t, \mathcal{D})$ is necessary to update. In other words, when minor updates occur and $\hat{f}(\mathbf{x}, t, \mathcal{D})$ is still accurate enough, we ignore them. To check the accuracy of $\hat{f}(\mathbf{x}, t, \mathcal{D})$, we update the labels of all validation data, and re-test the mean absolute error (MAE) of $\hat{f}(\mathbf{x}, t, \mathcal{D})$. If the difference between the original MAE and the new one is no larger than a predefined threshold δ_U, we do not update our model. Otherwise, we adopt an incremental learning approach as follows. First, we update the labels in the training and the validation data to reflect the update in the database. Second, we continue training our model with the updated training data until the validation error (MAE) does not increase in 3 consecutive epochs. Here the training does not start from scratch but from the current model. We incrementally train our model with all the training data to prevent catastrophic forgetting.

6 DISCUSSIONS

6.1 Model Complexity Analysis

We assume an FFN has hidden layers $\mathbf{a}_1, \ldots, \mathbf{a}_n$. The complexity of an FFN with input $\mathbf{x}$ and output $\mathbf{y}$ is $|\text{FFN}(\mathbf{x}, \mathbf{y})| = |\mathbf{x}| \cdot |\mathbf{a}_1| + \sum_{i=1}^{n-1} |\mathbf{a}_i| \cdot |\mathbf{a}_{i+1}| + |\mathbf{a}_n| \cdot |\mathbf{y}|$.

Our model contains three components: AE, FFN, and M. The complexity of AE is $|\text{FFN}(\mathbf{x}, \mathbf{z})|$. The complexity of FFN is $|\text{FFN}([\mathbf{x}; \mathbf{z}], \mathbf{t})|$, where $\mathbf{t}$ is the L-dimensional vector after Norm_{l_2}. Component M consists of an FFN and $(L+2)$ linear transformations. Its complexity is $|\text{FFN}([\mathbf{x}; \mathbf{z}], \mathbf{H})| + (L+2) \cdot |\mathbf{h}_i| + (L+2)$, where $\mathbf{H} = [\mathbf{h}_0; \ldots; \mathbf{h}_{L+1}]$. Thus, the final model complexity is $|\text{FFN}(\mathbf{x}, \mathbf{z})| + |\text{FFN}([\mathbf{x}; \mathbf{z}], \mathbf{t})| + |\text{FFN}([\mathbf{x}; \mathbf{z}], \mathbf{H})| + (L+2) \cdot |\mathbf{h}_i| + (L+2)$.

6.2 Comparison with Lattice Regression

Lattice regression models are the latest deep learning architectures for monotonic regression. We provide a comparison between ours and them applied to selectivity estimation. In order to perform an analytical comparison, we make the following simplification: (1) We assume that $\mathbf{x}$ and $\mathcal{D}$ are fixed so the selectivity only depends on t. (2) We consider a shallow version of DLN with one layer of calibrator and one layer of a single lattice.

With the above simplification, the DLN can be analytically represented as: $\hat{f}_{\text{DLN}}(t) = h(g(t; \mathbf{w}); \theta_0, \theta_1)$, where $g : t \in [0, t_{\max}] \mapsto z \in [0, 1]$ and $h(z; \theta_0, \theta_1) = (1-z)\theta_0 + z\theta_1$. Hence it degenerates to fitting a linear interpolation in a latent space. There is little learning for the function h, as its two parameters θ_0 and θ_1 are determined by the minimum and maximum selectivity values in the training data. Thus, the workhorse of the model is to learn the non-linear mapping of g. The calibrator also uses piecewise linear functions with L control points equivalent to our $(\tau_i, p_i)_{i=1}^{L}$. However, τ_is are equally spaced between 0 and $t_{\max}$, and only p_is are learnable. This design is not flexible for many value functions; e.g., if the function values change rapidly within a small interval, the calibrator will not adaptively allocate more control points to this area. We show this with 8 control points for both models to learn the function $y = f(t) = \frac{1}{10}\exp(t)$, $t \in [0, 10]$. The training data are 80 $(t_i, f(t_i))$ pairs where t_is are uniformly sampled in $[0, 10]$. We plot both models' estimation curves and their learned control points in Figure 3. The z values at the control points of DLN are shown on the right side of Figure 3(a). We observe: (1) The calibrator virtually determines the estimation as $h()$ degenerates to a simple scaling. (2) The calibrator's control points are evenly spaced in t, while our model learns to place more controls points in the "interesting area", i.e., where y values change rapidly. (3) As a result, our model approximates the value function much better than DLN.

Further, for DLN, the non-linear mapping on t is independent of $\mathbf{x}$ (even though we do not model $\mathbf{x}$ here). Even in the full-fledged DLN model, the calibration is performed on each input dimension independently. The full-fledged DLN model is too complex to analyze, so we only study it in our empirical evaluation. Nonetheless, we believe that the above inherent limitations still remain. Our empirical evaluation will also show that query-dependent fitting of the value function is critical in our problem and very effective for our model. Apart from DLN, recent studies also employ lattice regression and/or piecewise linear functions for learned index [22, 26]. Like DLN, their control points are also query independent, albeit not equally spaced.

6.3 Comparison with Clenshaw-Curtis Quadrature

Clenshaw-Curtis quadrature is able to approximate the integral $\int_0^{\tau_{max}} \hat{g}(\mathbf{x}, t, \mathcal{D})\text{d}t$, where $\hat{g} = \frac{\partial \hat{f}(\mathbf{x}, t, \mathcal{D})}{\partial t}$ in our problem. UMNN [41] is a recent work that adopts the idea to solve the autoregressive flow problem, and uses a neural network to model $\hat{g}$. In the Clenshaw-Curtis scheme [28], the cosine transform of $\hat{g}(cos\theta)$ is adopted and the discrete finite cosine transform is sampled at equidistant points $\theta = \frac{\pi s}{N}$, where $s = 1, \ldots, N$, and N is the number of sample points. Similar to DLN, it adopts the same integral approximation for different queries and ignores that integral points should depend on $\mathbf{x}$. In contrast, our method addresses this issue by using a query-dependent model, thereby delivering more flexibility.

7 EVALUATIONS

7.1 Experimental Settings

Datasets. We use three embedding datasets.

- **fasttext** is a pretrained word embedding dataset consisting of 1 million 300-dimensional vectors [1]. We evaluate Euclidean distance and cosine similarity and dub them fasttext-cos and fasttext-l_2, respectively.

- **face** is the MS-Celeb face image dataset [12]. We randomly extract 2 million images and obtain 128-dimensional embedding vectors using the faceNet model [32]. We only evaluate cosine similarity and dub the setting face-cos because vectors are already normalized.
- **YouTube** is a collection of video records and consists of 0.35 million vectors with 1770 dimensions [2]. We only evaluate cosine similarity and dub the setting YouTube-cos because vectors are already normalized.

Given a dataset $\mathcal{D}$, we randomly sampled 0.25 million vectors from $\mathcal{D}$ as query objects. We consider two settings to generate thresholds: (1) For the default setting, like the approach in [27], we sample a geometric sequence (by uniformly sampling from logarithmic values) of 40 selectivity values in the range $[1, 1\%|\mathcal{D}|]$ and calculate the corresponding thresholds; e.g., the cosine similarity for $1\%|\mathcal{D}|$ is around 0.5 on fasttext. (2) For the alternative setting, we generate cosine similarity thresholds in $[0.2, 1]$ using the beta distribution $\alpha = 3$ and $\beta = 2.5$, in order to simulate the case when people are more interested in thresholds in the middle. The selectivity is up to around $10\%|\mathcal{D}|$ on fasttext.

The resulting query workload, denoted by $\mathcal{Q}$, was uniformly split in 8:1:1 (by query objects) into training set $\mathcal{T}_{train}$, validation set $\mathcal{T}_{valid}$, and test set $\mathcal{T}_{test}$, in line with [40]. This ensures that none of the test query objects has been seen by the model during training or validation. Note that labels (i.e., true selectivities) were computed on $\mathcal{D}$, not $\mathcal{Q}$. For each setting, we tested on 5 sampled workloads to mitigate the effect of sampling error.

Methods. We compare the following approaches [3].

- **LSH** [44] is based on importance sampling using locality-sensitive hashing. It only works for cosine similarity due to the use of SimHash [4].
- **KDE** [27] is based on adaptive kernel density estimation for metric distance functions. To cope with cosine similarity, we normalize data to unit vectors and run KDE for Euclidean distance.
- **LightGBM** [39] is based on gradient boosting tree. We compare with the standard one (LightGBM) and the one with monotonicity enforced (LightGBM-m). XGBoost [5] has the same mechanism with LightGBM and was shown to deliver similar accuracy to LightGBM but in slower speed [40], so we do not compare it.
- Deep regression models: **DNN**, a vanilla feed-forward network; **MoE** [33], a mixture of expert model with sparse activation; **RMI** [23], a hierarchical mixture of expert model that achieves competitive performance on indexing for range queries; and **CardNet** [40], a feature extraction model followed by a regression model based on incremental prediction (we enable the accelerated estimation [40]).
- Lattice regression models: We adopt **DLN** [48] in this category.
- Clenshaw-Curtis quadrature model: We adopt **UMNN** [41].
- Our model is dubbed **SelNet**. The default setting of L is 50 and K is 3. δ_U for incremental learning is 20. We also evaluate two *ablated* models: (1) **SelNet$_{\text{-ct}}$** is SelNet without the cover tree partitioning, and (2) **SelNet$_{\text{-ad-ct}}$** is SelNet$_{\text{-ct}}$ without the query-dependent feature for control points (disabled by feeding a constant vector into the FFN that generates the τ vector).

[3] Please see Appendix B for model settings.

Table 1: Accuracy on fasttext-cos.

Model	MSE ($\times 10^5$)		MAE ($\times 10^2$)		MAPE	
	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$
LSH *	70.03	71.45	12.28	13.17	1.38	1.40
KDE *	64.13	64.22	11.71	11.72	1.03	1.01
LightGBM	133.22	130.07	8.44	8.47	0.98	0.97
LightGBM-m *	157.25	160.11	9.02	9.08	0.92	0.91
DNN	46.52	46.61	10.23	10.26	0.92	0.93
MoE	57.40	79.83	7.16	7.17	1.35	1.37
RMI	22.34	21.76	7.65	7.63	0.91	0.92
CardNet *	**15.85**	**16.12**	**5.34**	**5.39**	**0.89**	**0.84**
DLN *	56.22	58.57	10.66	10.54	1.07	1.06
UMNN *	23.22	24.69	6.67	6.71	1.21	1.22
SelNet *	**4.95**	**5.08**	**2.95**	**2.96**	**0.63**	**0.61**

Table 2: Accuracy on fasttext-l_2.

Model	MSE ($\times 10^5$)		MAE ($\times 10^2$)		MAPE	
	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$
KDE *	**24.34**	31.40	7.25	8.33	1.51	1.74
LightGBM	164.91	98.77	10.20	9.56	1.25	1.04
LightGBM-m *	171.46	101.22	10.13	9.84	1.22	1.20
DNN	46.22	63.54	10.24	11.25	1.28	1.33
MoE	558.56	725.74	8.08	8.31	1.16	1.22
RMI	31.12	33.16	7.94	8.02	0.98	0.99
CardNet *	24.53	**25.67**	**6.14**	**6.16**	**0.89**	**0.90**
DLN *	76.04	77.50	11.53	11.56	1.52	1.53
UMNN *	43.11	43.04	8.90	8.89	1.36	1.35
SelNet *	**7.65**	**7.87**	**3.51**	**3.56**	**0.75**	**0.76**

Table 3: Accuracy on face-cos.

Model	MSE ($\times 10^5$)		MAE ($\times 10^2$)		MAPE	
	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$
LSH *	277.82	277.35	24.44	25.36	1.25	1.28
KDE *	179.76	182.34	19.65	19.76	0.99	1.01
LightGBM	108.57	101.29	9.76	9.51	0.46	0.45
LightGBM-m *	112.44	114.62	10.43	10.46	0.48	0.49
DNN	196.56	110.77	20.05	17.14	0.87	0.89
MoE	15.66	21.25	**4.27**	4.32	0.29	0.30
RMI	26.53	27.22	6.86	6.87	0.37	0.39
CardNet *	**14.21**	**13.67**	4.33	**4.24**	**0.28**	**0.27**
DLN *	126.85	112.36	18.34	18.02	0.98	0.97
UMNN *	16.32	16.75	4.68	4.70	0.38	0.36
SelNet *	**5.03**	**4.96**	**2.46**	**2.43**	**0.22**	**0.23**

Error Metrics. We evaluate Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE), in line with [40]. We also report Mean Absolute Error (MAE).

Environment. Experiments were run on a server with an Intel Xeon E5-2640 @2.40GHz CPU and 256GB RAM, running Ubuntu 16.04.4 LTS. Models were implemented in Python and Tensorflow.

7.2 Comparison with State-of-the-art Methods

We report the accuracies of the competitors in Tables 1 – 4, where models that guarantee consistency are marked with *, and best and runner-up values are marked in boldface.

Table 4: Accuracy on YouTube-cos.

Model	MSE ($\times 10^4$)		MAE ($\times 10^2$)		MAPE	
	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$
LSH *	30.25	28.54	1.85	1.83	0.78	0.76
KDE *	28.76	29.25	1.89	1.90	0.74	0.70
LightGBM	39.50	40.11	1.99	2.00	0.51	0.52
LightGBM-m *	43.83	54.26	2.12	2.19	0.64	0.65
DNN	25.90	27.76	1.71	1.77	0.49	0.51
MoE	16.13	15.78	1.60	1.59	0.54	0.53
RMI	15.90	17.71	1.58	1.62	0.56	0.55
CardNet *	**12.42**	**14.12**	**1.42**	**1.44**	**0.48**	**0.48**
DLN *	29.25	29.37	1.91	1.92	0.70	0.69
UMNN *	19.04	20.57	1.64	1.69	0.50	0.49
SelNet *	**7.21**	**7.23**	**1.12**	**1.13**	**0.38**	**0.36**

Table 5: Empirical monotonicity (%) on face-cos.

LSH *	KDE *	LightGBM	LightGBM-m *	DNN	
100	100	86.34	100	78.22	
MoE	RMI	CardNet *	DLN *	UMNN *	SelNet *
94.82	90.48	100	100	100	100

Our model, SelNet, robustly and significantly outperforms existing models. It achieves substantial error reduction against the best of state-of-the-art methods, in all the three error metrics and all the settings. Compare to the runner-up model in each setting, the improvement is 2.0 – 3.3 times in MSE, 1.3 – 1.8 times in MAE, and 1.2 – 1.4 times in MAPE on test data.

We examine each category of models. We start with the sampling-based methods. KDE works better than LSH in most settings. In fact, KDE's performance even outperforms some deep learning regression based methods in a few cases (e.g., MSE in Table 2). For tree-based models, LightGBM and LightGBM-m do not perform well, and LightGBM-m cannot beat LightGBM in most cases. This indicates that the monotonic constraint, albeit better interpretability, decreases the performance of regression. Among the deep learning models (except ours), CardNet is generally the best thanks to its incremental prediction for each threshold interval. MoE is not good at large selectivities (MSE in Table 2). The performance of DLN is mediocre. The main reason is analyzed in Section 6.2. The accuracy of UMNN, which uses the same integral points for different queries, though better than DLN, still trails behind ours by a large margin.

7.3 Consistency Test

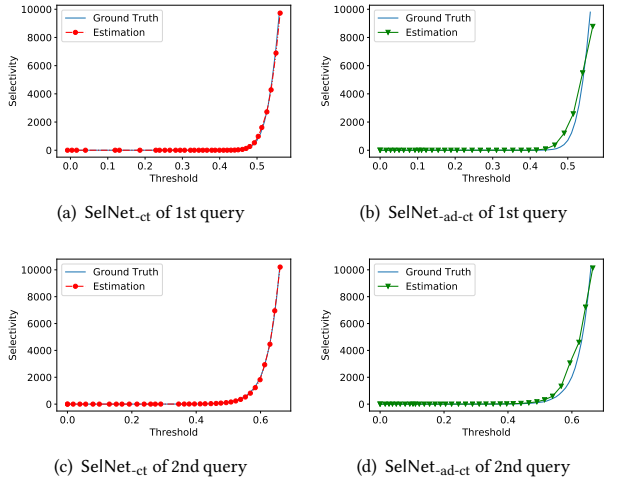
We compute the empirical monotonicity measure [7] and show the results in Table 5. The measure is the percentage of estimated pairs that violate the monotonicity, averaged over 200 queries. For each query, we sampled 100 thresholds, which form $\binom{100}{2}$ pairs. A low score indicates more inconsistent estimates. As expected, models without consistency guarantee cannot produce 100% monotonicity.

7.4 Ablation Study

SelNet$_{-ct}$ v.s. SelNet. Table 6 shows that the partitioning improves the performance, especially on fasttext-l_2, where 1.6 times MSE improvement is observed on test data. This is because: each model deals with a subset of the dataset for better fit; and the ground truth label values for each model are reduced, which makes it easier to fit our piecewise linear function with the same number of control points, as the value function is less steep.

Table 6: Ablation study.

Dataset	Model	MSE ($\times 10^5$)		MAE ($\times 10^2$)		MAPE	
		$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$
fasttext-cos	SelNet	**4.95**	**5.08**	**2.95**	**2.96**	**0.63**	**0.61**
	SelNet$_{-ct}$	5.79	5.83	3.12	3.11	0.65	0.63
	SelNet$_{-ad-ct}$	14.92	14.79	5.22	5.18	1.23	1.22
fasttext-l_2	SelNet	**7.65**	**7.87**	**3.51**	**3.56**	**0.75**	**0.76**
	SelNet$_{-ct}$	13.21	12.63	4.33	4.37	0.82	0.81
	SelNet$_{-ad-ct}$	39.77	39.59	8.77	8.72	2.93	2.90
face-cos	SelNet	**5.03**	**4.96**	**2.46**	**2.43**	**0.22**	**0.23**
	SelNet$_{-ct}$	5.38	5.31	2.81	2.92	0.23	0.24
	SelNet$_{-ad-ct}$	16.87	16.02	4.71	4.65	0.38	0.37
YouTube-cos	SelNet	**7.21**	**7.23**	**1.12**	**1.13**	**0.38**	**0.36**
	SelNet$_{-ct}$	9.04	9.07	1.22	1.20	0.41	0.39
	SelNet$_{-ad-ct}$	15.27	16.54	1.57	1.59	0.52	0.53



(a) SelNet$_{-ct}$ of 1st query (b) SelNet$_{-ad-ct}$ of 1st query

(c) SelNet$_{-ct}$ of 2nd query (d) SelNet$_{-ad-ct}$ of 2nd query

Figure 4: Control points on fasttext-cos.

SelNet$_{-ct}$ v.s. SelNet$_{-ad-ct}$. The difference between the two models is whether control points are dependent on $\mathbf{x}$ or not. Table 6 shows this feature has a significant impact on accuracy across all the settings and all the error metrics. When query-dependent control points are employed to fit the selectivity curve, the improvements in MSE, MAE, and MAPE are up to 3.1, 2.0, and 3.6 times, respectively. To further illustrate the difference, we plot in Figure 4 the control points learned by the two models for two randomly picked queries. We can see that SelNet$_{-ad-ct}$ uses the same set of control points for the two queries. This is obviously not ideal – SelNet$_{-ad-ct}$ fails to fit the ground truth curve closely, especially for quickly changing selectivity values, and this results in larger errors across all the three metrics. In contrast, SelNet$_{-ct}$ devotes more points to the threshold intervals in which the selectivity changes rapidly.

7.5 Estimation Time

Table 7 shows the estimate time on the testing data. We also report the time of running a state-of-the-art selection algorithm [20] to obtain the exact cardinality (referred to as SimSelect). All the models except LSH are at least on order of magnitude faster than SimSelect.

Table 7: Average estimation time (milliseconds).

Model	fasttext-cos	fasttext-l_2	face-cos	YouTube-cos
SimSelect	7.45	8.14	9.65	6.11
LSH *	1.59	-	1.08	2.35
KDE *	0.68	0.79	0.59	0.94
LightGBM	0.29	0.28	0.18	0.52
LightGBM-m *	0.28	0.28	0.19	0.50
DNN	**0.07**	**0.07**	**0.03**	**0.16**
MoE	0.33	0.36	0.27	0.49
RMI	0.36	0.34	0.25	0.47
CardNet *	0.20	0.19	0.14	0.31
DLN *	0.81	0.83	0.65	1.22
UMNN *	0.37	0.39	0.24	0.52
SelNet *	0.34	0.35	0.24	0.51

Table 8: Training time (hours).

Model	fasttext-cos	fasttext-l_2	face-cos	YouTube-cos
KDE *	**1.1**	**1.1**	**0.7**	**0.8**
LightGBM	1.6	1.6	1.2	1.0
LightGBM-m *	1.5	1.4	1.2	1.1
DNN	2.9	2.9	2.7	2.5
MoE	4.3	4.2	3.8	3.7
RMI	4.6	4.5	3.9	3.5
CardNet *	3.5	3.5	3.1	2.9
DLN *	6.7	6.6	6	5.7
UMNN *	5.3	5.2	4.9	4.4
SelNet *	5.1	5.0	4.7	4.6

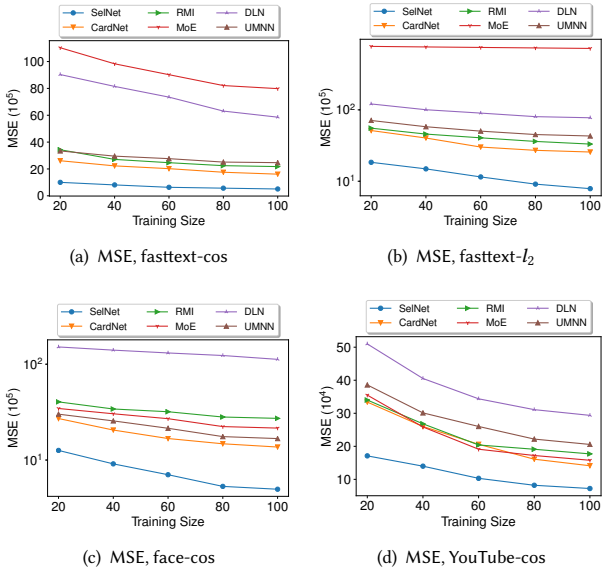
Our model is on a par with other deep learning models (except DNN), and faster than LSH and KDE. Although DNN is very fast due to its simple model structure, its accuracy is much worse than ours, e.g., by 22 times of MSE on face-cos.

7.6 Training

Table 8 shows the training times of the competitors. As expected, traditional learning models are faster to train. Our models spend around 5 hours, similar to other deep models. In Figure 5, we show the performances, measured by MSE, of the deep learning models by varying the scale of training examples from 20% to 100% of the original training data. All the models perform worse with fewer training data, but our models are more robust, showing moderate accuracy loss.

7.7 Data Update

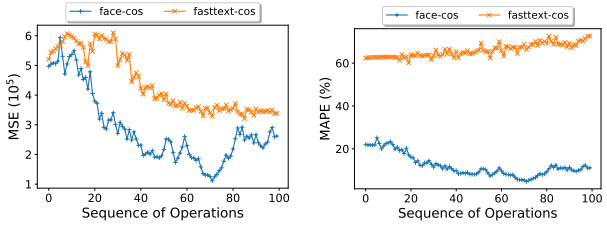
We generate a stream of 100 update operations, each with an insertion or deletion of 5 records on face-cos and fasttext-cos, to evaluate our incremental learning technique. Figure 6 plots how MSE and MAPE change with the stream. The general trend is that the error is decreasing when there are more updates, except for MAPE on fasttext-cos, which keeps almost the same. The result indicates that incremental learning is able to keep up with the updated data. Besides, SelNet only spends 1.5 – 2.0 minutes for each incremental learning, showcasing its speed to cope with updates.



(a) MSE, fasttext-cos (b) MSE, fasttext-l_2

(c) MSE, face-cos (d) MSE, YouTube-cos

Figure 5: Varying training data size.



(a) MSE (b) MAPE

Figure 6: Data update.

Table 9: Varying number of control points on fasttext-l_2.

Error Metric	Number of Control Points			
	10	50	90	130
MSE on $\mathcal{T}_{test}$ ($\times 10^5$)	13.06	7.65	7.93	10.47
MAE on $\mathcal{T}_{test}$ ($\times 10^2$)	4.85	3.51	3.56	3.92
MAPE on $\mathcal{T}_{test}$	0.87	0.75	0.76	0.79

7.8 Evaluation of Hyper-parameters

Table 9 shows the accuracy when we vary the number of control points L on fasttext-l_2. A small value leads to underfitting towards the curve of thresholds, while a large value increases the learning difficulty. $L = 50$ achieves the best performance.

Table 10 reports the accuracy when we vary the partition size K on fasttext-l_2. There is no partitioning when $K = 1$. We observe that the partitioning is useful, but the improvement is small when partition size exceeds 3, and estimation time also substantially increases. This means a small partition size ($K = 3$) suffices to achieve good performance. For partitioning strategy, we compare cover tree partitioning (CT) with random partitioning (RP) and k-means

Table 10: Varying partition size on fasttext-l_2.

Error Metric	Partition Size			
	1	3	6	9
MSE on $\mathcal{T}_{test}$ ($\times 10^5$)	13.21	7.65	6.82	6.75
MAE on $\mathcal{T}_{test}$ ($\times 10^2$)	4.33	3.51	3.36	3.11
MAPE on $\mathcal{T}_{test}$	0.82	0.75	0.77	0.74
Estimation Time (ms)	0.16	0.36	0.79	1.24

Table 11: Varying partitioning method on fasttext-l_2.

Error Metric	CT (3)	RP (3)	KM (3)
MSE on $\mathcal{T}_{test}$ ($\times 10^5$)	7.87	8.02	9.14
MAE on $\mathcal{T}_{test}$ ($\times 10^2$)	3.56	3.57	3.64
MAPE on $\mathcal{T}_{test}$	0.76	0.78	0.79

Table 12: Accuracy on fasttext-cos (thresholds follow beta distribution $B(3, 2.5)$).

Model	MSE ($\times 10^8$)		MAE ($\times 10^3$)		MAPE	
	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$	$\mathcal{T}_{valid}$	$\mathcal{T}_{test}$
LSH *	74.79	69.88	14.55	14.42	1.37	1.35
KDE *	69.53	69.18	14.00	13.90	0.99	1.01
LightGBM	57.44	59.28	12.42	12.48	0.77	0.73
LightGBM-m *	68.86	70.11	13.87	13.92	0.74	0.76
DNN	36.10	36.04	10.04	10.03	0.87	0.89
MoE	12.39	11.64	4.33	4.25	1.26	1.20
RMI	13.32	14.15	4.87	4.89	0.62	0.64
CardNet *	6.75	6.21	**3.42**	**3.31**	**0.44**	**0.42**
DLN *	48.26	47.89	11.50	11.39	1.11	1.13
UMNN *	**6.20**	**6.09**	3.58	3.54	0.55	0.52
SelNet *	**1.62**	**1.64**	**1.74**	**1.73**	**0.38**	**0.38**

partitioning (KM) in Table 11. CT delivers the best performance. KM is the worst because it tends to cause imbalance in the partition.

7.9 Alternative Threshold Setting

We evaluate the case when thresholds are generated by the alternative setting of beta distribution. Table 12 reports the accuracy on fasttext-cos. Compared with the default setting (Table 1), all the models report larger MSE and MAE (note MSE is measured in $\times 10^8$ and MAE in $\times 10^3$ here), but most models performs better in MAPE. This is because the selectivities in this setting (in $[1, 10\%|\mathcal{D}|]$) are usually larger than the default one (in $[1, 1\%|\mathcal{D}|]$), but the variance (when selectivities are normalized) is smaller. SelNet is still consistently better than the others: on test data, it improves MSE by 3.7 times, MAE by 2.0 times, and MAPE by 1.1 times over the runner-up model in each setting. This demonstrates that our model is robust against distribution change in thresholds.

8 CONCLUSION

In this paper, we tackled the selectivity estimation problem for high-dimensional data using a deep learning architecture. Our method is based on learning monotonic, query-dependent piece-wise linear function. This provides the flexibility of our model to approximate the selectivity curve while guaranteeing the consistency of estimation. We proposed a partitioning technique to cope with large-scale datasets and an incremental learning technique for updates. Our experiments demonstrated that the proposed model outperforms state-of-the-art methods significantly across a range of settings including datasets, distance functions, and error metrics.

REFERENCES

[1] https://fasttext.cc/docs/en/english-vectors.html.
[2] http://www.cs.tau.ac.il/~wolf/ytfaces/index.html.
[3] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000.
[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.
[5] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.
[6] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.
[7] H. Daniels and M. Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.
[8] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446, 2017.
[9] 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.
[10] J. Fox. Robust regression: Appendix to an r and s-plus companion to applied regression, 2002.
[11] E. Garcia and M. Gupta. Lattice regression. In *NIPS*, pages 594–602, 2009.
[12] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao. MS-Celeb-1M: A dataset and benchmark for large scale face recognition. In *ECCV*, 2016.
[13] 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.
[14] Q. Han, T. Wang, S. Chatterjee, and R. J. Samworth. Isotonic regression in general dimensions. *arXiv preprint arXiv:1708.09468*, 2017.
[15] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep learning models for selectivity estimation of multi-attribute queries. In *SIGMOD*, pages 1035–1050, 2020.
[16] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, GPU-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, pages 1477–1492, 2015.
[17] P. J. Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.
[18] Y. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.
[19] M. Izbicki and C. R. Shelton. Faster cover trees. In F. R. Bach and D. M. Blei, editors, *ICML*, pages 1162–1170, 2015.
[20] M. Izbicki and C. R. Shelton. Faster cover trees. In *ICML*, pages 1162–1170, 2015.
[21] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR*, 2019.
[22] A. Kipf, R. Marcus, A. van Renen, M. Stoian, A. Kemper, T. Kraska, and T. Neumann. Radixspline: a single-pass learned index. In *aiDM@SIGMOD*, pages 5:1–5:5, 2020.
[23] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.
[24] M. Kula. Metadata embeddings for user and item cold-start recommendations. *CoRR*, abs/1507.08439, 2015.
[25] S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud. A comprehensive analysis of deep regression. *arXiv preprint arXiv:1803.08450*, 2018.
[26] P. Li, H. Lu, Q. Zheng, L. Yang, and G. Pan. LISA: A learned index structure for spatial data. In *SIGMOD*, pages 2119–2133, 2020.
[27] M. Mattig, T. Fober, C. Beilschmidt, and B. Seeger. Kernel-based cardinality estimation on metric data. In *EDBT*, pages 349–360, 2018.
[28] M. Novelinkova. Comparison of clenshaw-curtis and gauss quadrature. In *WDS*, volume 11, pages 67–71, 2011.
[29] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. An empirical analysis of deep learning for cardinality estimation. *CoRR*, abs/1905.06425, 2019.
[30] Y. Park, S. Zhong, and B. Mozafari. Quicksel: Quick selectivity learning with mixture models. In *SIGMOD*, pages 1017–1033, 2020.
[31] L. Prunty. Curve fitting with smooth functions that are piecewise-linear in the limit. *Biometrics*, pages 857–866, 1983.
[32] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823, 2015.
[33] 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.
[34] J. Spouge, H. Wan, and W. Wilbur. Least squares isotonic regression in two dimensions. *Journal of Optimization Theory and Applications*, 117(3):585–605, 2003.

[35] J. Sun and G. Li. An end-to-end learning-based cost estimator. *PVLDB*, 13(3):307–319, 2019.
[36] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, pages 3476–3483, 2013.
[37] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, pages 1653–1660, 2014.
[38] B. Walenz, S. Sintos, S. Roy, and J. Yang. Learning to sample: Counting with complex queries. *Proc. VLDB Endow.*, 13(3):390–402, 2019.
[39] D. Wang, Y. Zhang, and Y. Zhao. Lightgbm: An effective mirna classification method in breast cancer patients. In *ICCBB*, pages 7–11, 2017.
[40] Y. Wang, C. Xiao, J. Qin, X. Cao, Y. Sun, W. Wang, and M. Onizuka. Monotonic cardinality estimation of similarity selection: A deep learning approach. In *SIGMOD*, pages 1197–1212, 2020.
[41] A. Wehenkel and G. Louppe. Unconstrained monotonic neural networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *NeurIPS*, pages 1543–1553, 2019.
[42] K.-Y. Whang, S.-W. Kim, and G. Wiederhold. Dynamic maintenance of data distribution for selectivity estimation. *The VLDB Journal*, 3(1):29–51, 1994.
[43] W. Wu, J. F. Naughton, and H. Singh. Sampling-based query re-optimization. In *SIGMOD*, pages 1721–1736, 2016.
[44] X. Wu, M. Charikar, and V. Natchu. Local density estimation in high dimensions. In *ICML*, pages 5293–5301, 2018.
[45] Y. Wu, D. Agrawal, and A. El Abbadi. Query estimation by adaptive sampling. In *ICDE*, pages 639–648, 2002.
[46] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, P. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep unsupervised cardinality estimation. *Proc. VLDB Endow.*, 13(3):279–292, 2019.
[47] H. Ying, F. Zhuang, F. Zhang, Y. Liu, G. Xu, X. Xie, H. Xiong, and J. Wu. Sequential recommender system based on hierarchical attention networks. In *IJCAI*, pages 3926–3932, 2018.
[48] S. You, D. Ding, K. Canini, J. Pfeifer, and M. Gupta. Deep lattice networks and partial monotonic functions. In *NIPS*, pages 2981–2989, 2017.

A PROOF

Lemma 1.

Proof. Assume $t \in [\tau_{i-1}, \tau_i)$, then $t+\epsilon$ is in $[\tau_{i-1}, \tau_i)$ or $[\tau_i, \tau_{i+1})$. In the first case, $\hat{f}(\mathbf{x}, t+\epsilon, \mathcal{D}; \Theta) - \hat{f}(\mathbf{x}, t, \mathcal{D}; \Theta) = \frac{\epsilon}{\tau_i - \tau_{i-1}} \cdot (p_i - p_{i-1}) \geq 0$. In the second case, $\hat{f}(\mathbf{x}, t, \mathcal{D}; \Theta) \leq p_i$ and $\hat{f}(\mathbf{x}, t+\epsilon, \mathcal{D}; \Theta) \geq p_i$. Therefore, $\hat{f}(\mathbf{x}, t, \mathcal{D}; \Theta)$ is non-decreasing in t. □

B EXPERIMENT SETUP

B.1 Model Settings

Hyperparameter and training settings are given below.

- DNN is a vanilla FFN with four hidden layers of sizes 512, 512, 512, and 256.
- MoE consists of 30 expert models, each an FFN with three hidden layers of sizes 512, 512, and 512. We used top-3 experts for the prediction.
- RMI has three levels, with 1, 4, and 8 models, respectively. Each model is an FFN with four hidden layers with sizes 512, 512, 512, and 256.
- DLN is an architecture of six layers: calibrators, linear embedding, calibrators, ensemble of lattices, calibrators, and linear embedding.
- UMNN is an FFN with four hidden layers of sizes 512, 512, 512 and 256 to implement the derivative. $\frac{\partial f(\mathbf{x},t,\mathcal{D})}{\partial t}$. $f(\mathbf{x}, t, \mathcal{D})$ is computed by Clenshaw-Curtis quadrature with learned derivatives.
- SelNet: We use an FFN with two hidden layers to estimate $\boldsymbol{\tau}$, and an FFN in Equation 9 with four hidden layers to estimate $\mathbf{p}$. The encoder and decoder of AE are implemented with an FFN with three hidden layers. For face-cos and YouTube-cos, the sizes of the first three (or two, if it only has two) hidden layers of the three FFNs are 512, and the sizes of all the other hidden layers are 256. For fasttext-cos and fasttext-l_2, the sizes of the first hidden layer of the these FFNs are 1024, and the others remain the same as above. The number of control parameters L is 50. The default partition size K is 3. $t_{\max}$ is 54 for Euclidean distance. For cosine similarity, we equivalently convert it to Euclidean distance on unit vectors, and set $t_{\max}$ as 1. The learning rates of face-cos, fasttext-cos, fasttext-l_2 and YouTube-cos are 0.00003, 0.00002, 0.00002 and 0.00003. $|\mathbf{h}_i|$ $(0 \leq i \leq L+1)$ in model M is 100. The batch size is 512 for all datasets. We train all models in 1500 epochs, and select the ones with the smallest validation errors. For training with data partitioning, we use $T = 300$ and $\beta = 0.1$. δ_U for incremental learning is 20.

For LSH and KDE, we use 2,000 samples to keep the estimation cost reasonable. For all the other models, we train them with the same Huber loss on the logarithm of the ground truth and prediction; all hyper-parameters are fine-tuned according to the validation set. DNN, MoE and RMI cannot directly handle the threshold t. We learn a non-linear transformation of t into an m dimensional embedding vector, i.e., $\mathbf{t} = \text{ReLU}(\mathbf{w}t)$. Then we concatenate it with $\mathbf{x}$ as the input to these models.

B.2 Evaluation Metric

We evaluate Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). They are defined as:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2,$$

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}_i - y_i|,$$

$$\text{MAPE} = \frac{1}{m}\sum_{i=1}^{m}\left|\frac{\hat{y}_i - y_i}{y_i}\right|,$$

where y_i is the ground truth value and $\hat{y}_i$ is the estimated value.