

# Consistent and Flexible Selectivity Estimation for High-Dimensional Data

Yaoshu Wang<sup>1</sup>, Chuan Xiao<sup>2,3</sup>, Jianbin Qin<sup>1</sup>, Rui Mao<sup>1</sup>, Makoto Onizuka<sup>2</sup>, Wei Wang<sup>4,5</sup>, Rui Zhang<sup>6</sup>, and Yoshiharu Ishikawa<sup>3</sup>

<sup>1</sup>Shenzhen Institute of Computing Sciences, Shenzhen University, <sup>2</sup>Osaka University, <sup>3</sup>Nagoya University, <sup>4</sup>Dongguan University of Technology, <sup>5</sup>University of New South Wales, <sup>6</sup>www.ruizhang.info  
yaoshuw@sics.ac.cn, {chuanx, onizuka}@ist.osaka-u.ac.jp, {qinjianbin, mao}@szu.edu.cn, weiw@cse.unsw.edu.au, rui.zhang@ieee.org, ishikawa@i.nagoya-u.ac.jp

## 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 distance function and query object, 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 consistently outperforms state-of-the-art models in accuracy in an efficient way and is useful for real applications.

## CCS CONCEPTS

• **Information systems** → **Query optimization**; *Entity resolution*; • **Computing methodologies** → *Neural networks*.

## KEYWORDS

selectivity estimation; high-dimensional data; piecewise linear function; deep neural network

## 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  $dist(\cdot, \cdot)$ , and a distance threshold  $t$ , estimate the number of objects  $\mathbf{o}$  in a database that satisfy  $dist(\mathbf{x}, \mathbf{o}) \leq t$ . This problem is also known as local density estimation [54] or spherical range counting [9] in theoretical computer science. It is an essential procedure in density estimation in statistics [52] and density-based outlier detection [11] in data mining. For example, for text analysis, one may want to determine the popularity of a topic; for e-commerce, an analyst may want to find out if a user/item is an outlier; for clustering, the algorithm may converge faster if we start with seeds in denser regions. In the database area, accurate estimation helps to find an optimal query execution plan in

databases dealing with high-dimensional data [24]. Hands-off entity matching systems [16] extract paths from random forests and take each path – a conjunction of similarity predicates over multiple attributes (e.g., “ $EU(\text{name}) \leq 0.25$  AND  $EU(\text{affiliations}) \leq 0.4$  AND  $EU(\text{research interests}) \leq 0.45$ ”, where  $EU(\cdot)$  measures the Euclidean distance between word embeddings) – as a blocking rule, and efficient blocking can be achieved if we find a good query execution plan [50]. In addition, many text or image retrieval systems resort to distributed representations. Given a query, a similarity selection is often invoked to obtain a set of candidates to be further verified by sophisticated models. Estimating the number of candidates helps to estimate the overall query processing time an end-to-end system to create a service level agreement.

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 [26], 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 distance function and 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 [15] or limiting to multi-linear functions [17]. To improve the accuracy

W. Wang, R. Mao and J. Qin are the joint corresponding authors.

on large-scale datasets, we propose a partition-based method to divide the database into disjoint subsets and learn a local model on each of them. Since update may exist in the database, we employ incremental learning to cope with this issue.

We perform experiments on six real datasets. The results show that our method outperforms various state-of-the-art models. Compared to the best existing model [50], the improvement of accuracy is up to 5 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, robust against update in the database, and useful in estimating overall query processing time in a semantic search application.

## 2 RELATED WORK

*Traditional Estimation Models.* Selectivity estimation has been extensively studied in database systems, where prevalent approaches are based on sampling [53, 55], histograms [26], or sketches [14]. However, few of them are applicable to high-dimensional data due to data sparsity or the curse of dimensionality. For cosine similarity, Wu *et al.* [54] proposed to use locality-sensitive hashing (LSH) as a means of importance sampling to tackle data sparsity. Kernel density estimation (KDE) [24, 36] has been proposed to handle selectivity estimation in metric space. Mattig *et al.* [36] proposed to alleviate 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 a regression problem with query object and threshold as input features, if the consistent requirement is not enforced. A representative approach is quantized regression [7, 8]. Recent trend uses deep regression models. Vanilla deep regression [32, 46, 47] learns good representations of input patterns. The mixture of expert model (MoE) [43] 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) [31] 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) [29, 45] in relational databases, amid a set of recent advances in learning methods for this task [23, 38, 39, 48, 56]. They target SQL queries where each predicate involves one attribute. [23, 56] employ autoregressive models. [39] only deals with low dimensionality. [29, 38, 45] become a deep neural network if we regard a vector as an attribute.

*Models with Consistency Guarantee.* Gradient boosting trees (e.g., XGBoost [13] and LightGBM [49]) support monotonic regression. Lattice regression [17, 19, 21, 57] uses a multi-linearly interpolated lookup table for regression. By enforcing constraints on its parameter values, it can guarantee monotonicity. To accommodate high dimensional inputs, Fard *et al.* [17] proposed to build an ensemble of lattice using subsets of input features. Deep lattice network (DLN) [57] 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 [33]. UMNN [51] is an autoregressive flow model which adopts Clenshaw-Curtis quadrature to achieve monotonicity. Other monotonic models include isotonic regression [22, 44] and MinMaxNet [15]. CardNet [50] 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  $\tau$ , and then predicts the selectivity for distance  $[0, 1, \dots, \tau]$  respectively with  $(\tau + 1)$  encoder-decoder models. When applying to high-dimensional data, it has the following drawbacks: the mapping from the input threshold to  $\tau$  is not injective, i.e., multiple thresholds may be mapped to the same  $\tau$  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  $d$ -dimensional vectors  $\mathcal{D} = \{\mathbf{o}_i\}_{i=1}^n$ ,  $\mathbf{o}_i \in \mathbb{R}^d$ , a distance function  $\text{dist}(\cdot, \cdot)$ , a scalar threshold  $t$ , and a query object  $\mathbf{x} \in \mathbb{R}^d$ , estimate the selectivity in the database, i.e.,  $|\{\mathbf{o} \mid \text{dist}(\mathbf{x}, \mathbf{o}) \leq t, \mathbf{o} \in \mathcal{D}\}|$ .

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. In addition, the query object does not have to be in the database, and we do not make any assumption on the distance function, meaning that the function does not have to be metric.

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: instead of estimating one function  $f$ , we estimate multiple functions such that each function's output range is a small fraction of the selectivity  $y$ . For example, suppose  $y = y_1 + y_2$  and  $t = t_1 + t_2$ . If  $y_1$  and  $y_2$  are approximately linear in  $[0, t_1]$  and  $(t_1, t_2]$ , respectively, but with different slopes, then we can use two linear models for the two threshold ranges. We may also exploit this idea and divide  $y$  with disjoint subsets of  $\mathcal{D}$ . Hence 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:  $[\tau_0, \tau_1, \dots, \tau_{L+1}]$  such that  $\tau_i < \tau_j$  if  $i < j$ ,  $\tau_0 = 0$ , and  $\tau_{L+1} = t_{\max} + \epsilon$ , where  $\epsilon$  is a small positive quantity<sup>1</sup>. Let  $g_i(\mathbf{x}, t)$  be an interpolant function for interval  $[\tau_{i-1}, \tau_i)$ . Then we have

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

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

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

## 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  $[\tau_0, \tau_1, \dots, \tau_{L+1}]$  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  $\tau_i$  values are called *control points*. Given a query object  $\mathbf{x}$ , let  $p_i$  denote the estimated selectivity for a threshold  $\tau_i$ . For the  $g_i$  function in Eq. (1), we have

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

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

Using the above design for  $\Theta$  has the following property to guarantee the consistency<sup>2</sup>.

**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 [40]. 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  $L$   $\tau_i$  values and the  $(L+2)$   $p_i$  values are the parameters to be learned. We use the expected loss between  $f$  and  $\hat{f}$ :

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

<sup>1</sup> $\epsilon$  is used to cover the corner case of  $t = t_{\max}$  in Eq. (1).

<sup>2</sup>Proof is provided in Appendix A.

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 [25] applied to the logarithmic values of  $y$  and  $\hat{y}$ . To prevent numeric errors, we also pad the input by a small positive quantity  $\epsilon$ . Let  $r \stackrel{\text{def}}{=} \ln(y + \epsilon) - \ln(\hat{y} + \epsilon)$ . Then

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

$\delta$  is set to 1.345, the standard recommended value [18]. 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  $\ell_2$  loss, it encourages the model to fit large selectivities well, and if we use  $\ell_1$  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 database, and this favors deep learning which often requires large training sets.

In our model,  $\tau_i$  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  $\tau_i$ s and  $p_i$ s, followed by the overall neural network architecture.

*Control Points ( $\tau_i$ s).* We learn the increments between  $\tau_i$ s.

$$\tau_i(\mathbf{x}) = \sum_{j=0}^{i-1} \Delta_{\tau}(\mathbf{x})[j], \quad (4)$$

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

$\text{Norm}_{l_2}$  is a normalized squared function defined as

$$\text{Norm}_{l_2}(\mathbf{t}) = \left[ \frac{t_1^2 + \frac{\epsilon}{L}}{\mathbf{t}^T \mathbf{t} + \epsilon}, \dots, \frac{t_L^2 + \frac{\epsilon}{L}}{\mathbf{t}^T \mathbf{t} + \epsilon} \right],$$

where  $\epsilon$  is a small positive quantity to avoid dividing by zero, 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 vector  $\boldsymbol{\tau} = [0; \tau_1; \tau_2; \dots; \tau_L; t_{\max}]$ .

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)  $\text{Softmax}$  aims to highlight the important part rather than partitioning  $\mathbf{t}$ , while our goal is to rationally partition the range  $[0, t_{\max}]$  into several intervals such that the piecewise linear function can fit well.

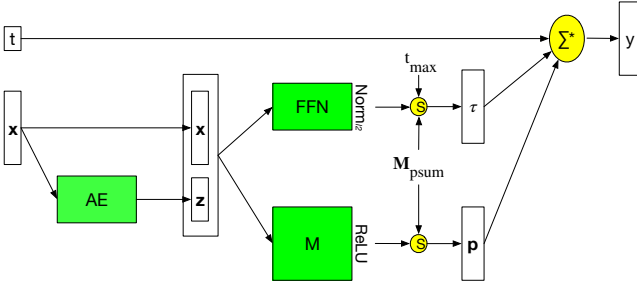


Figure 1: Network architecture.

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

$$p_i(x) = \sum_{j=0}^i \Delta_p(x)[j], \quad (6)$$

$$\text{where } \Delta_p(x) = \text{ReLU}(g^{(p)}(x)). \quad (7)$$

Then we have a vector  $\mathbf{p} = [p_0; p_1; \dots; p_{L+1}]$ . Here, we learn  $(L + 1)$  increments  $(p_i - p_{i-1})$  instead of directly learning  $(L + 2)$   $p_i$ s. 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.* Figure 1 shows our network architecture.

The input  $\mathbf{x}$  is first transformed to  $\mathbf{z}$ , a latent representation obtained by an autoencoder (AE). 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  $J_{\text{AE}}$ ):

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

$\mathbf{x}$  is concatenated with  $\mathbf{z}$ , 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  $t_{\text{max}}$  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  $\Sigma^*$  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.

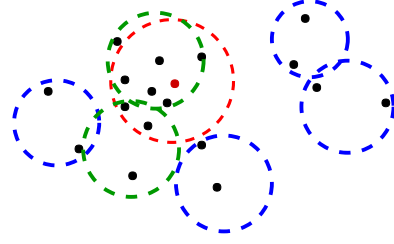


Figure 2: Data partitioning by cover tree.

*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:

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

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:

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

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.

*Partitioning Method.* We utilize a cover tree [27] 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  $K$  ( $K' \leq K$ ) 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

$f_c : (\mathbf{x}, t) \rightarrow \{0, 1\}^K$  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 and modify  $f_c$  as  $f_c : (\mathbf{x}, t) \rightarrow \{1\}^K$ .

*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.

## 5.4 Dealing with Data Updates

When the database  $\mathcal{D}$  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  $\delta_U$ , 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  $\mathbf{a}_1, \dots, \mathbf{a}_n$ . 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  $\text{Norm}_{l_2}$ . 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  $\mathbf{H} = [\mathbf{h}_0; \dots; \mathbf{h}_{L+1}]$ . Thus, the final model complexity is  $|\text{FFN}(\mathbf{x}, \mathbf{z})| + |\text{FFN}([\mathbf{x}; \mathbf{z}], \mathbf{t})| + |\text{FFN}([\mathbf{x}; \mathbf{z}], \mathbf{H})| + (L+2) \cdot |\mathbf{h}_i| + (L+2)$ .

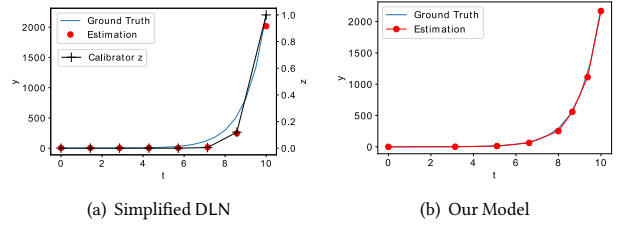


Figure 3: Comparison of simplified DLN and our model.

## 6.2 Comparison with Other Models

*Lattice Regression.* Lattice regression models [17, 19, 21, 57] are the latest deep learning architectures for monotonic regression. We provide a comparison between ours and them applied to selectivity estimation. For the sake of an analytical comparison, we assume  $\mathbf{x}$  and  $\mathcal{D}$  are fixed so the selectivity only depends on  $t$ , and consider a shallow version of DLN [57] 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  $\theta_0$  and  $\theta_1$  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,  $\tau_i$ s are equally spaced between 0 and  $t_{\max}$ , and only  $p_i$ s 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  $t_i$ s 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  $h(\cdot)$  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. Apart from DLN, recent studies also employ lattice regression and/or piecewise linear functions for learned index [30, 33]. Like DLN, their control points are also query independent, albeit not equally spaced.

**Table 1: Statistics of datasets.**

Dataset	Source	Domain	# Objects	Dimensionality	Distance
fastText	[1]	text	1M	300	Euclidean
GloVe	[2]	text	1.9M	300	Euclidean
MS-Celeb	[20]	image	2M	128	cosine
YouTube	[3]	video	0.35M	1770	cosine
DEEP	[4]	image	100M	96	cosine
SIFT	[5]	image	200M	128	cosine

*Clenshaw-Curtis Quadrature.* Clenshaw-Curtis quadrature [37] is able to approximate the integral  $\int_0^{\tau_{max}} \hat{g}(\mathbf{x}, t, \mathcal{D}) dt$ , where  $\hat{g} = \frac{\partial \hat{f}(\mathbf{x}, t, \mathcal{D})}{\partial t}$  in our problem. UMNN [51] is a recent work that adopts the idea to solve the autoregressive flow problem, and uses a neural network to model  $\hat{g}$ . In [37], 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  $s = 1, \dots, N$ , 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.

*Query-Driven Quantized Regression.* The main idea of query-driven quantized regression [7, 8] is to quantize the query space and find prototypes (the closest one or multiple related ones) for the given query object. Then the output space is quantized by prototypes, and localized regressions are used to estimate the selectivity for corresponding prototypes. Like our model, they also employ a query-dependent design. The differences from ours are: (1) [7, 8] divide the query space of  $(\mathbf{x}, t)$  while we divide the range of threshold  $t$  using  $\mathbf{x}$ . (2) The number of prototypes is finite and often up to thousands in [7, 8], while our model chooses the selectivity curve for the query object via an FFN and model  $M$  (Figure 1), which yield an unlimited number of curves in  $\mathbb{R}^{L+2}$ . (3) We employ deep regression for higher accuracy. (4) We directly partition the database  $\mathcal{D}$  and train multiple deep models to deal with the subsets of  $\mathcal{D}$  that may differ in data distribution, while the data subspace in [8] is defined by its query prototype.

## 7 EVALUATIONS

### 7.1 Experimental Settings

*Datasets.* We use six datasets. The statistics is given in Table 1. We preprocess MS-Celeb by faceNet [42] to obtain vectors. The other datasets have already been transformed to high-dimensional data. GloVe, YouTube, DEEP, and SIFT were also used in previous work [50] or nearest neighbor search benchmarks [10, 34].

We randomly sample 0.25M vectors from each dataset  $\mathcal{D}$  as query objects.

The resulting query workload, denoted by  $\mathcal{Q}$ , was uniformly split in 8:1:1 (by query objects) into training, validation, and test sets. So none of the test query objects has been seen by the model during training or validation. Note that labels (i.e., true selectivities) are computed on  $\mathcal{D}$ , not  $\mathcal{Q}$ . For each training query object, we iterate through all the generated thresholds and add them to the training set. We randomly choose 3 generated thresholds for each validation or test query object. Due to the large number of training data,

we randomly select training instances for each batch instead of continuously loading them, and the training procedure terminates when the mean squared error of the validation set does not increase in 5 consecutive epochs. For each setting, we tested on 5 sampled workloads to mitigate the effect of sampling error.

*Methods.* We compare the following approaches<sup>3</sup>.

- **RS** is a random sampling approach. For each query, we uniformly sample  $0.1\%|\mathcal{D}|$  objects for the first four datasets and  $0.01\%|\mathcal{D}|$  objects for DEEP and SIFT. Then we use `scipy.spatial.distance.cdist` to compute the distances to the query objects in a batch manner.
- **IS** [54] is an importance sampling approach using locality-sensitive hashing. It only works for cosine similarity due to the use of SimHash [12]. We enforce monotonicity by using deterministic sampling w.r.t. the query object.
- **KDE** [36] 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.
- **QR-1** [7] and **QR-2** [8] are two query-driven quantized regression models. We use the linear model in [7] for QR-1.
- **LightGBM** [49] is based on gradient boosting decision trees (CARTs). Each rule in a CART is in the form of  $x_i < a$  ( $x_i$  is the  $i$ -th dimension of  $\mathbf{x}$ ) or  $t < b$ .
- Deep regression models: **DNN**, a vanilla feed-forward network; **MoE** [43], a mixture of expert model with sparse activation; **RMI** [31], a hierarchical mixture of expert model; and **CardNet** [50], a regression model based on incremental prediction (we enable the accelerated estimation [50]).
- Lattice regression models: We adopt **DLN** [57] in this category.
- Clenshaw-Curtis quadrature model: We adopt **UMNN** [51].
- Our model is dubbed **SelNet**<sup>4</sup>. The default setting of  $L$  (number of control points) is 50 and  $K$  (partition size) is 3. The predefined threshold  $\delta_U$  for incremental learning is 20. We also evaluate two *ablated* models: (1) **SelNet<sub>-ct</sub>** is SelNet without the cover tree partitioning, and (2) **SelNet<sub>-ad-ct</sub>** is SelNet<sub>-ct</sub> without the query-dependent feature for control points (disabled by feeding a constant vector into the FFN that generates the  $\tau$  vector).

*Error Metrics.* We evaluate Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and 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 Accuracy

We report accuracies in Table 2, where monotonic models are marked with \*, and best values are marked in boldface. Our model, SelNet, consistently 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 on each dataset, the improvement is 2.0 – 5.0 times in MSE, 1.3 – 3.3 times in MAE, and 1.2 – 1.7 times in MAPE, and is more significant on larger datasets.

We examine each category of models. We start with the sampling-based methods. KDE works better than RS and IS in most settings.

<sup>3</sup>Please see Appendix B for model settings.

<sup>4</sup>The source code is available at [6].

**Table 2: Accuracy (MSE and MAE measured in  $10^5$  and  $10^2$ , respectively).**

Model	fastText			GloVe			MS-Celeb			YouTube			DEEP			SIFT		
	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE
RS *	22.38	7.45	1.40	34.85	8.71	1.09	28.64	8.09	1.30	2.95	1.89	0.88	84732.32	725.21	1.26	30317437.60	9496.55	0.98
IS *	-	-	-	-	-	-	104.58	14.25	1.25	2.85	1.83	0.76	50242.10	314.12	0.97	32049511.88	10612.42	0.92
KDE *	21.46	6.57	1.28	37.52	8.93	0.87	36.43	8.42	1.02	2.93	1.90	0.70	39497.24	298.10	0.85	27651109.31	9581.89	0.91
QR-1	41.79	8.95	1.02	50.13	9.21	0.99	74.35	11.60	1.06	3.42	2.01	0.71	37054.11	292.16	0.78	28077423.18	9953.34	0.94
QR-2	34.35	8.03	0.97	42.47	9.01	0.86	42.94	9.05	0.89	2.74	1.85	0.55	31485.85	273.22	0.67	22048511.27	8542.73	0.71
LightGBM	98.77	9.56	1.04	72.11	10.87	0.89	101.29	9.51	0.45	4.01	2.00	0.52	44036.45	301.88	0.85	23849121.36	9005.17	0.75
DNN	63.54	11.25	1.33	52.31	9.39	0.91	110.77	17.14	0.89	2.78	1.77	0.51	20454.11	192.13	0.69	24465910.26	7144.68	0.55
MoE	45.90	8.50	0.91	30.14	7.05	0.91	21.25	4.32	0.30	1.58	1.59	0.53	18068.93	170.51	0.65	14750194.30	6327.47	0.40
RMI	26.16	6.10	0.87	29.32	6.89	0.74	22.16	6.07	0.35	1.77	1.62	0.55	9498.21	116.54	0.67	8906108.00	4650.29	0.42
CardNet *	25.67	6.16	0.90	27.05	6.19	0.78	13.67	4.08	0.27	1.41	1.44	0.48	9230.48	117.37	0.67	7248693.54	4851.43	0.40
DLN *	77.50	11.56	1.53	52.26	10.27	0.89	82.35	11.85	0.97	2.94	1.92	0.69	58291.42	353.08	0.94	23059384.16	8058.49	0.51
UMNN *	33.26	7.20	0.92	33.50	7.98	0.86	16.75	4.70	0.36	2.06	1.69	0.49	10603.68	131.04	0.73	10201332.32	5443.31	0.43
SelNet *	<b>7.87</b>	<b>3.56</b>	<b>0.76</b>	<b>9.17</b>	<b>3.83</b>	<b>0.68</b>	<b>4.96</b>	<b>2.43</b>	<b>0.23</b>	<b>0.72</b>	<b>1.13</b>	<b>0.36</b>	<b>2243.42</b>	<b>51.92</b>	<b>0.51</b>	<b>1464247.70</b>	<b>1406.63</b>	<b>0.23</b>

**Table 3: Empirical monotonicity (%) on MS-Celeb.**

RS *	IS *	KDE *	QR-1	QR-2
100	100	100	85.39	84.86
LightGBM	DNN	MoE	RMI	
86.34	78.22	94.82	90.48	
CardNet *	DLN *	UMNN *	SelNet *	
100	100	100	100	

In fact, KDE’s performance even outperforms some deep learning regression based methods in a few cases (e.g., MSE on fastText). Among non-deep learning models, there is no best model across all the datasets, though QR-2 prevails on more datasets than others. Among the deep learning models other than ours, CardNet is generally the best thanks to its incremental prediction for each threshold interval. 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 [15] and show the results in Table 3. 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

Table 4 shows that the partitioning (SelNet v.s. SelNet<sub>ct</sub>) improves MSE, MAE, and MAPE by up to 3.4, 2.1, and 1.2 times, respectively, and the effect is more remarkable on large datasets. 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. Using query-dependent control points (SelNet<sub>ct</sub> v.s. SelNet<sub>ad-ct</sub>) also has a significant impact on accuracy across all the settings and all the error metrics. The improvements in MSE, MAE, and MAPE are up to 3.1, 2.0, and 3.6 times, respectively.

**Table 4: Ablation study.**

Dataset	Model	MSE ( $\times 10^5$ )	MAE ( $\times 10^2$ )	MAPE
fastText	SelNet	<b>7.87</b>	<b>3.56</b>	<b>0.76</b>
	SelNet <sub>ct</sub>	12.63	4.37	0.81
	SelNet <sub>ad-ct</sub>	39.59	8.72	2.90
GloVe	SelNet	<b>9.17</b>	<b>3.83</b>	<b>0.68</b>
	SelNet <sub>ct</sub>	22.43	5.82	0.70
	SelNet <sub>ad-ct</sub>	32.59	6.92	0.90
MS-Celeb	SelNet	<b>4.96</b>	<b>2.43</b>	<b>0.23</b>
	SelNet <sub>ct</sub>	5.31	2.92	0.24
	SelNet <sub>ad-ct</sub>	16.02	4.65	0.37
YouTube	SelNet	<b>0.72</b>	<b>1.13</b>	<b>0.36</b>
	SelNet <sub>ct</sub>	0.90	1.20	0.39
	SelNet <sub>ad-ct</sub>	1.65	1.59	0.53
DEEP	SelNet	<b>2243.42</b>	<b>51.92</b>	<b>0.51</b>
	SelNet <sub>ct</sub>	5861.43	72.18	0.58
	SelNet <sub>ad-ct</sub>	9012.57	101.42	0.71
SIFT	SelNet	<b>1464247.70</b>	<b>1406.63</b>	<b>0.23</b>
	SelNet <sub>ct</sub>	4958113.22	2911.86	0.27
	SelNet <sub>ad-ct</sub>	6904808.25	3855.53	0.54

### 7.5 Estimation Time

Table 5 reports the estimation times of the competitors. We also report the time of running a state-of-the-art selection algorithm (CoverTree [27]) to obtain the exact selectivity. All the models except IS are at least one order of magnitude faster than CoverTree, and the gaps increase to three orders of magnitude on DEEP and SIFT. Our model is on a par with other deep learning models (except DNN) and faster than sampling and quantized regression methods.

### 7.6 Training

Table 6 shows the training times. Non-deep models are faster to train. Our models spend 5 – 6 hours, similar to other deep models. In Figure 4, 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 fastText and MS-Celeb, to evaluate

**Table 5: Average estimation time (milliseconds).**

Model	fastText	GloVe	MS-Celeb	YouTube	DEEP	SIFT
CoverTree	8.14	8.85	9.65	6.11	214	395
RS *	0.46	0.51	0.49	0.52	2.54	4.72
IS *	-	-	1.08	2.35	4.97	6.81
KDE *	0.79	0.68	0.59	0.94	1.48	2.05
QR-1	0.86	0.98	0.97	1.03	2.21	2.82
QR-2	0.79	0.99	0.95	1.10	2.32	2.91
LightGBM	0.28	0.30	0.18	0.52	0.26	0.26
DNN	<b>0.07</b>	<b>0.10</b>	<b>0.03</b>	<b>0.16</b>	<b>0.11</b>	<b>0.10</b>
MoE	0.36	0.33	0.27	0.49	0.29	0.33
RMI	0.34	0.38	0.25	0.47	0.27	0.30
CardNet *	0.19	0.26	0.14	0.31	0.22	0.28
DLN *	0.83	0.69	0.65	1.22	0.64	0.80
UMNN *	0.39	0.32	0.24	0.52	0.26	0.32
SelNet *	0.35	0.31	0.24	0.51	0.29	0.36

**Table 6: Training time (hours).**

Model	fastText	GloVe	MS-Celeb	YouTube	DEEP	SIFT
KDE *	<b>1.1</b>	<b>1.5</b>	<b>0.7</b>	<b>0.8</b>	2.5	3.6
QR-1	1.8	2.1	1.5	1.2	3.9	4.6
QR-2	1.5	2.0	1.6	1.2	3.6	4.7
LightGBM	2.1	1.9	2.2	2.1	<b>1.9</b>	<b>2.1</b>
DNN	2.9	2.1	2.8	2.9	2.5	2.9
MoE	4.9	5.4	4.9	4.7	4.3	4.4
RMI	5.4	5.6	4.8	5.3	4.6	4.8
CardNet *	3.8	3.9	3.3	3.2	3.5	3.8
DLN *	6.9	7.1	6.0	6.5	6.3	6.4
UMNN *	5.5	5.7	4.9	5.2	5.4	4.6
SelNet *	6.0	5.5	5.2	5.6	5.2	5.0

**Table 7: Varying number of control points on fastText.**

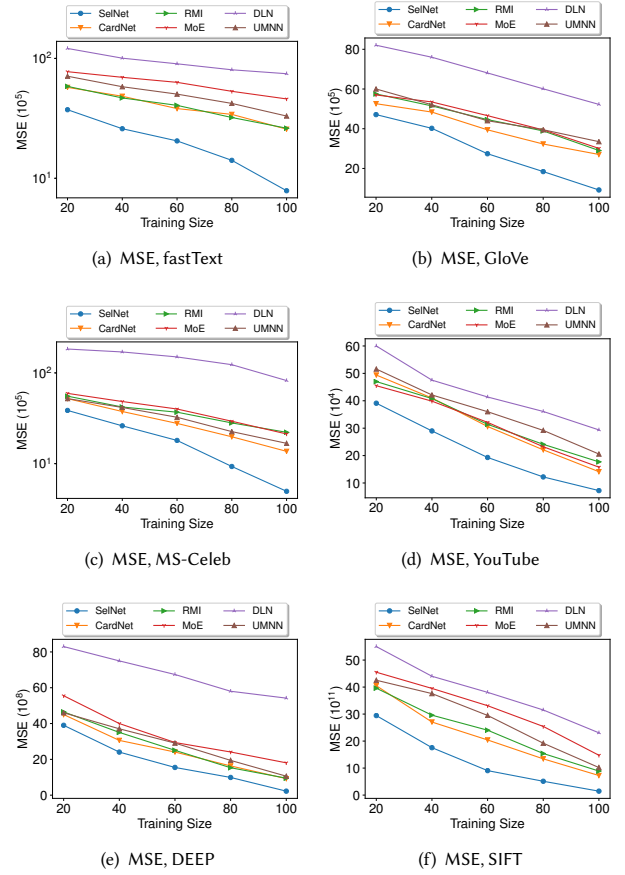
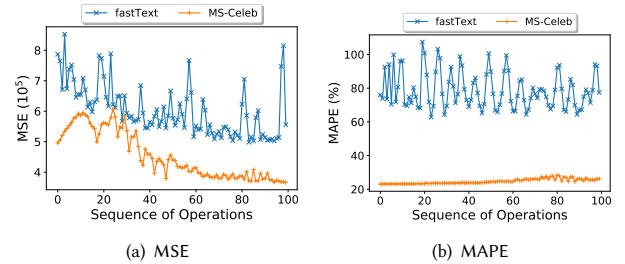
Error Metric	Number of Control Points			
	10	50	90	130
MSE ( $\times 10^3$ )	13.06	7.87	7.93	10.47
MAE ( $\times 10^2$ )	4.85	3.56	3.56	3.92
MAPE	0.87	0.76	0.76	0.79

our incremental learning technique. Figure 5 plots how MSE and MAPE change with the stream. The general trend is that the MSE is decreasing when there are more updates, while MAPE fluctuates or keeps almost the same. Such difference is caused by the change of labels (i.e., true selectivities) in the stream. Nonetheless, 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.

## 7.8 Evaluation of Hyper-Parameters

Table 7 shows the accuracy when we vary the number of control points  $L$  on fastText. 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 8 reports the accuracy when we vary the partition size  $K$  on fastText. 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


**Figure 4: Varying training data size.**

**Figure 5: Data update.**

good performance. For partitioning strategy, we compare cover tree partitioning (CT) with random partitioning (RP) and  $k$ -means partitioning (KM) in Table 9. CT delivers the best performance. KM is the worst because it tends to cause imbalance in the partition.

## 7.9 Generalizability

To show the generalizability of our model, we evaluate the performance on the queries that significantly differ from the records in the training data. To prepare such queries, we first perform a

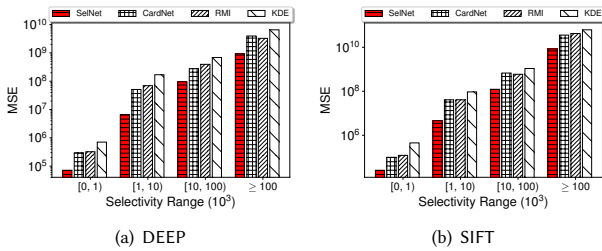


**Table 8: Varying partition size on fastText.**

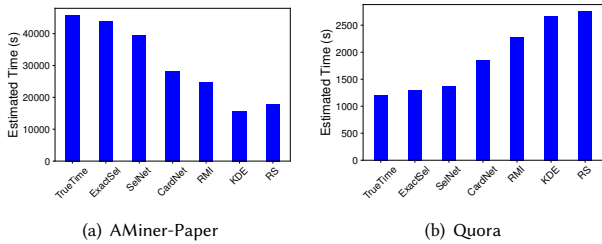
Error Metric	Partition Size			
	1	3	6	9
MSE ( $\times 10^5$ )	12.63	7.87	6.82	6.75
MAE ( $\times 10^2$ )	4.37	3.56	3.36	3.11
MAPE	0.81	0.76	0.77	0.74
Estimation Time (ms)	0.16	0.35	0.79	1.24

**Table 9: Varying partitioning method on fastText.**

Error Metric	CT (3)	RP (3)	KM (3)
MSE ( $\times 10^5$ )	7.87	8.02	9.14
MAE ( $\times 10^2$ )	3.56	3.57	3.64
MAPE	0.76	0.78	0.79



**Figure 6: Generalizability.**



**Figure 7: Estimated search time (10,000 queries).**

$k$ -means clustering on  $\mathcal{D}$ . We randomly sample 10,000 query objects from  $\mathcal{D}$  (excluding the queries used for training) and add Gaussian noise [58]. Then we pick the top-2,000 ones having the largest sum of squared distance to the  $k$  centroids. Figure 6 show the performances of KDE, RMI, CardNet, and SelNet on DEEP and SIFT, measured by MSE. The queries are grouped by selectivity range. In each selectivity group, SelNet consistently outperforms the other models, and the advantage is around one order of magnitude. This result demonstrates that our model generalizes well for out-of-dataset queries.

### 7.10 Performance in Semantic Search

To evaluate the usefulness of SelNet, we consider estimating the overall processing time for a query workload of semantic search: given a query text entry, we want to find matching records in the database. Estimating the query processing time may help to create a service level agreement. We use two datasets, AMiner-Paper publications (2.1M records) and Quora questions (0.8M records). AMiner-Paper has four attributes: title, authors, venue, and year.

We follow [35] and concatenate attribute names and values as one string. Quora has one attribute. Then we embed each record to a 768-dimensional vector by Sentence-BERT [41].

10,000 records are sampled from each dataset as queries. To process a query, we first embed it by Sentence-BERT [41], and then use Faiss [28] to find candidate records whose cosine similarity to the query embedding is no less than 0.9. The candidates are verified using DITTO [35]. Hence the overall query processing time can be estimated as:  $\text{avg\_Faiss\_time} \times 10000 + \text{avg\_DITTO\_time} \times \text{Faiss\_recall} \times \sum_1^{10000} \text{estimated\_selectivity\_of\_query\_i}$ . The average times and Faiss recall are obtained by running a small query workload. For selectivity, we consider RS, KDE, RMI, CardNet, SelNet, and an oracle that outputs the exact selectivity (ExactSel).

We plot the estimated time of processing 10,000 queries in Figure 7, where TrueTime indicates the ground truth. The models tend to underestimate on AMiner-Paper and overestimate on Quora. SelNet’s high accuracy in selectivity estimation pays off. Compared to the ground truth, SelNet’s error is 13% on AMiner-Paper and 16% on Quora, close to ExactSel’s and much lower than the other models’ (at least 39%). SelNet is also efficient; e.g., running the workload to obtain the ground truth on AMiner-Paper spends 13 hours, which is twice the time of preparing training data + training SelNet + estimating for 10,000 queries. Seeing SelNet’s scalability in estimation time (Table 5) and training time (Table 6), we believe that the advantage will be more substantial on larger datasets.

## 8 CONCLUSION

We tackled the selectivity estimation problem for high-dimensional data. 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 showed the superiority of the proposed model in accuracy across a variety of datasets, distance functions, and error metrics. The experiments also demonstrated the usefulness of our model in a semantic search application.

**Acknowledgements** This work was supported by NSFC 62072311 and U2001212, Guangdong Basic and Applied Basic Research Foundation 2019A1515111047 and 2020B1515120028, Guangdong Peral River Recruitment Program of Talents 2019ZT08X603, JSPS Kak-ehi 16H01722, 17H06099, 18H04093, and 19K11979, and ARC DPs 170103710 and 180103411.

## REFERENCES

- [1] <https://fasttext.cc/docs/en/english-vectors.html>.
- [2] <https://nlp.stanford.edu/projects/glove/>.
- [3] <http://www.cs.tau.ac.il/~wolf/ytfaces/index.html>.
- [4] <http://sites.skoltech.ru/compvision/noimi/>.
- [5] <http://corpus-texmex.irisa.fr/>.
- [6] <https://github.com/yysls88/SelNet-Estimation>.
- [7] C. Anagnostopoulos and P. Triantafillou. Learning set cardinality in distance nearest neighbours. In *ICDM*, pages 691–696, 2015.
- [8] C. Anagnostopoulos and P. Triantafillou. Query-driven learning for predictive analytics of data subspace cardinality. *ACM Trans. Knowl. Discov. Data*, 11(4):47:1–47:46, 2017.
- [9] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate spherical range counting. In *SODA*, pages 535–544, 2005.
- [10] M. Aumüller, E. Bernhardsson, and A. J. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.*, 87, 2020.

- [11] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000.
- [12] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.
- [13] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.
- [14] 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.
- [15] H. Daniels and M. Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.
- [16] 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.
- [17] 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.
- [18] J. Fox. Robust regression: Appendix to an r and s-plus companion to applied regression, 2002.
- [19] E. Garcia and M. Gupta. Lattice regression. In *NIPS*, pages 594–602, 2009.
- [20] 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.
- [21] 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.
- [22] Q. Han, T. Wang, S. Chatterjee, and R. J. Samworth. Isotonic regression in general dimensions. *arXiv preprint arXiv:1708.09468*, 2017.
- [23] 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.
- [24] M. Heimerl, M. Kiefer, and V. Markl. Self-tuning, GPU-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, pages 1477–1492, 2015.
- [25] P. J. Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.
- [26] Y. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.
- [27] M. Izbicki and C. R. Shelton. Faster cover trees. In *ICML*, pages 1162–1170, 2015.
- [28] H. Jégou, thhjs Douze, and J. Johnson. Facebook ai similarity search (faiss). <https://github.com/facebookresearch/faiss>.
- [29] 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.
- [30] 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.
- [31] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, pages 489–504, 2018.
- [32] S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud. A comprehensive analysis of deep regression. *arXiv preprint arXiv:1803.08450*, 2018.
- [33] 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.
- [34] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement. *IEEE Trans. Knowl. Data Eng.*, 32(8):1475–1488, 2020.
- [35] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan. Deep entity matching with pre-trained language models. *PVLDB*, 14(1):50–60, 2020.
- [36] M. Mattig, T. Fober, C. Beilshmidt, and B. Seeger. Kernel-based cardinality estimation on metric data. In *EDBT*, pages 349–360, 2018.
- [37] M. Novelinkova. Comparison of clenshaw-curtis and gauss quadrature. In *WDS*, volume 11, pages 67–71, 2011.
- [38] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. An empirical analysis of deep learning for cardinality estimation. *CoRR*, abs/1905.06425, 2019.
- [39] Y. Park, S. Zhong, and B. Mozafari. Quickselect: Quick selectivity learning with mixture models. In *SIGMOD*, pages 1017–1033, 2020.
- [40] L. Prunty. Curve fitting with smooth functions that are piecewise-linear in the limit. *Biometrics*, pages 857–866, 1983.
- [41] N. Reimers and I. Gurevych. Sentence embeddings using siamese BERT-networks. In *EMNLP-IJCNLP*, pages 3980–3990, 2019.
- [42] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823, 2015.
- [43] 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.
- [44] 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.
- [45] J. Sun and G. Li. An end-to-end learning-based cost estimator. *PVLDB*, 13(3):307–319, 2019.
- [46] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, pages 3476–3483, 2013.
- [47] A. Toshev and C. Szegedy. DeepPose: Human pose estimation via deep neural networks. In *CVPR*, pages 1653–1660, 2014.
- [48] B. Walenz, S. Sintos, S. Roy, and J. Yang. Learning to sample: Counting with complex queries. *PVLDB*, 13(3):390–402, 2019.
- [49] D. Wang, Y. Zhang, and Y. Zhao. Lightgbm: An effective mirna classification method in breast cancer patients. In *ICCB*, pages 7–11, 2017.
- [50] 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.
- [51] A. Wehenkel and G. Louppe. Unconstrained monotonic neural networks. In *NeurIPS*, pages 1543–1553, 2019.
- [52] K.-Y. Whang, S.-W. Kim, and G. Wiederhold. Dynamic maintenance of data distribution for selectivity estimation. *VLDB J.*, 3(1):29–51, 1994.
- [53] W. Wu, J. F. Naughton, and H. Singh. Sampling-based query re-optimization. In *SIGMOD*, pages 1721–1736, 2016.
- [54] X. Wu, M. Charikar, and V. Natchu. Local density estimation in high dimensions. In *ICML*, pages 5293–5301, 2018.
- [55] Y. Wu, D. Agrawal, and A. El Abbadi. Query estimation by adaptive sampling. In *ICDE*, pages 639–648, 2002.
- [56] 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. *PVLDB*, 13(3):279–292, 2019.
- [57] S. You, D. Ding, K. Canini, J. Pfeifer, and M. Gupta. Deep lattice networks and partial monotonic functions. In *NIPS*, pages 2981–2989, 2017.
- [58] D. Zhang and Z. Yang. Word embedding perturbation for sentence classification. *arXiv preprint arXiv:1804.08166*, 2018.

## APPENDIX

### 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  $t$ .  $\square$

### B EXPERIMENT SETUP

#### B.1 Model Settings

Hyperparameter and training settings are given below.

- IS and KDE: The sample size is 2000.
- QR-1 and QR-2: The number of query prototypes is 2000.
- LightGBM: The number of CARTs is 1000.
- 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  $\tau$ , 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 MS-Celeb and YouTube, 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, GloVe, DEEP, and SIFT, the sizes of the first hidden layer of 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  $t_{\max} = 1$ . The learning rates of MS-Celeb, fastText, YouTube, GloVe, DEEP, and SIFT are 0.00003, 0.00002, 0.00003, 0.0001, 0.0001, and 0.0001, respectively.  $|\mathbf{h}_i|$  ( $0 \leq i \leq L+1$ ) in model  $M$  is 100. The batch size is 512 for all the datasets. We train all the models in 1500 epochs and select the ones with the smallest validation error. For training with data partitioning, we use  $T = 300$  and  $\beta = 0.1$ .  $\delta_U$  for incremental learning is 20.

For the learning models, we train them with the same Huber loss over the logarithms of the ground truth and the predicted value. All the hyper-parameters are fine-tuned to minimize the validation error. 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 Metrics

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.