Selectivity Estimation for Range Predicates using Lightweight Models

Anshuman Dutt[1], Chi Wang[1], Azade Nazi[2]*, Srikanth Kandula[1], Vivek Narasayya[1], Surajit Chaudhuri[1]
[1]Microsoft Research, [2]Google AI
[1]{andut,wang.chi,srikanth,viveknar,surajitc}@microsoft.com, [2]azade@google.com

ABSTRACT

Query optimizers depend on selectivity estimates of query predicates to produce a good execution plan. When a query contains multiple predicates, today's optimizers use a variety of assumptions, such as independence between predicates, to estimate selectivity. While such techniques have the benefit of fast estimation and small memory footprint, they often incur large selectivity estimation errors. In this work, we reconsider selectivity estimation as a regression problem. We explore application of neural networks and tree-based ensembles to the important problem of selectivity estimation of multi-dimensional range predicates. While their straightforward application does not outperform even simple baselines, we propose two simple yet effective design choices, i.e., regression label transformation and feature engineering, motivated by the selectivity estimation context. Through extensive empirical evaluation across a variety of datasets, we show that the proposed models deliver both highly accurate estimates as well as fast estimation.

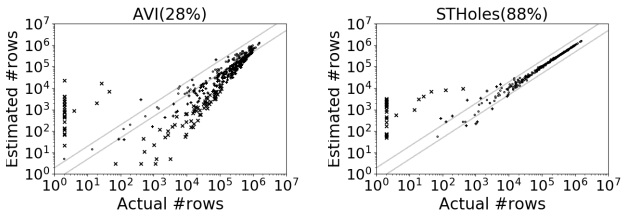
PVLDB Reference Format:
Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. Selectivity Estimation for Range Predicates using Lightweight Models. *PVLDB*, 12(9): 1044-1057, 2019.
DOI: https://doi.org/10.14778/3329772.3329780

1. INTRODUCTION

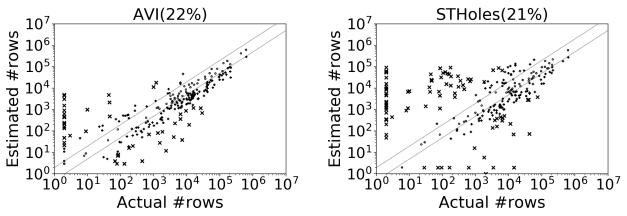
Query optimizers use selectivity estimates to identify a good execution plan. Ideally, a selectivity estimation technique should provide accurate and fast estimates, and use a data structure that has small memory footprint and is efficient to construct and maintain [19]. The requirement of fast estimation follows from the expectation that query optimization time should be small [13, 7], and is an important practical constraint in database systems.

To elaborate, current database systems primarily use histograms for selectivity estimation of predicates on single table attributes [5, 7]. For multi-dimensional predicates, the default method to calculate combined selectivity is based on a heuristic assumption such as attribute value independence (AVI) [4, 5, 7]. When attributes are correlated, estimates based on such assumptions may lead to huge errors resulting in low quality plans [27, 32]. Some database systems also provide an option for a small data sample [7]. While sampling can capture attribute correlations well, small samples have bad accuracy for selective predicates [34]. Nevertheless, database systems use such methods as they support fast selectivity estimation, e.g., multi-dimensional estimation based on AVI takes $< 100\mu$sec [38].

The research community has been actively working on the multi-dimensional selectivity estimation problem [19] leading to many variants of multi-dimensional histograms [12, 21, 38], techniques that use random samples [22], or use both histograms and samples [34] – Section 8 provides a review of related work. They improve accuracy at the cost of significant increase in space or time overhead, because they fundamentally rely on more histogram buckets or larger samples to capture the data distribution in a high-dimensional



(a) For 2-dimensional predicates over a highly correlated attribute-pair



(b) For 4-dimensional predicates over attributes with mixed degree of correlation

Figure 1: Estimation quality for AVI and STHoles. Plot title shows percentage of small errors [ratio-error < 2]

*The work was done while the author was at Microsoft Research

Proceedings of the VLDB Endowment, Vol. 12, No. 9
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3329772.3329780

space. Query-driven (self tuning) histograms [40, 10, 12, 22] use more histogram buckets in the subspace that is relevant for the current query workload, to improve the accuracy-cost trade-off in a targeted fashion. The accuracy of such techniques can degrade when workload queries have more intersections with each other and they spread across a large fraction of high-dimensional space.

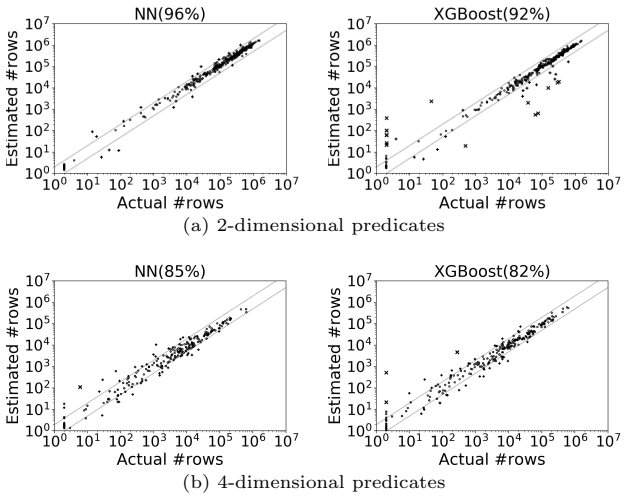
We analyzed estimates based on both kinds of histogram techniques: (i) AVI assumption, that uses only one dimensional histograms; (ii) STHoles [12], that uses a query driven multi-dimensional histogram. For this experiment, we used a representative real world dataset and a set of queries distributed all over the domain space. Figure 1 shows a scatter plot of actual number of rows satisfying the predicates (s) vs. corresponding estimated values ($\hat{s}$), for 2D and 4D predicates. Note that, queries with small estimation errors, i.e., $max\left(\frac{s}{\hat{s}}, \frac{\hat{s}}{s}\right) < 2$, lie in a *band-shaped area* along the diagonal. Queries with larger errors are farther away from this region. For AVI, only one-third of the queries have small errors and large errors are quite frequent. In contrast, STHoles provides highly accurate estimates for 2D predicates using only 200 buckets – except for very selective predicates ($s < 100$). For 4D predicates, STHoles resulted in large errors even with 3500 buckets and 10× larger estimation time.

In addition to above approaches, selectivity estimation has also been formulated as a *regression* problem: "Given a set of queries labeled with actual selectivity values, learn a function from a query to its selectivity". Such labeled queries can be collected as feedback from prior query executions [40, 14], as proposed by self-tuning approaches; or they can be generated offline in a data-driven fashion, similar to histogram construction (see Section 6). Past attempts with regression formulation typically employed neural networks [11, 26, 29, 28, 25]. These methods are not designed for fast estimation of multi-dimensional range selectivity.

1.1 Contributions

Regression models for range selectivity estimation. In this work, we study whether regression techniques can be used for accurate selectivity estimation of multi-dimensional range predicates, especially under the practical constraints of estimation time and memory footprint. Our study includes neural networks and tree-based ensembles. The choice of these techniques is governed by the following reasons: (1) they have an inherent ability to learn complex, non-linear functions; (2) recent work [9, 16, 24] has produced highly-optimized libraries based on these techniques.

Design choices for lightweight models. Notwithstanding the promise, our initial exploration with these techniques resulted in inferior accuracy compared to simple baselines like AVI. Although the learned models perform better as we increase the model size and complexity, we cannot afford arbitrarily large models. We propose two simple, yet effective design choices that improve accuracy of models without increasing model size. First, we use log-transformed version of regression labels. While the transformation is very simple, it has a strong impact on the effectiveness of regression, particularly because relative error is more relevant in the selectivity estimation context [32]. Second, we use a set of heuristic selectivity estimators as extra features in addition to predicate ranges. They help the model learn to differentiate between queries that have very similar range predicates but significantly different actual selectivities. Importantly, these estimators are computationally efficient. Both these design choices are generic and help both neural networks and tree-based ensembles.



(a) 2-dimensional predicates

(b) 4-dimensional predicates

Figure 2: Estimation quality for proposed regression models. Plot title shows the percentage of small errors [ratio-error < 2]

Experimental evaluation highlights. We perform extensive empirical evaluation across multiple real-world datasets to show that the proposed models provide fast and highly accurate estimates for multi-dimensional ranges compared to existing techniques including AVI, STHoles, uniform random sampling and a state-of-the-art kernel density based technique (KDE) [22] that improve over sampling.

Figure 2 shows the accuracy of proposed models based on neural network (NN) and XGBoost [16] (a tree-based ensemble), on the same 2D and 4D predicates as in Figure 1. Observe that for both regression techniques, the percentage of small errors is more than 80% and large errors are quite rare. The errors are small even for highly selective queries.

Not only our models achieve high accuracy, but also they have fast estimation time. The end-to-end estimation time for our models is between 1-2× of AVI, depending on whether the extra features are used. In both cases, the accuracy is much better than AVI. For comparison, STHoles/KDE could not beat the accuracy of our best models even with 10× larger estimation time.

We use datasets with up to 10 dimensions to demonstrate that the desirable properties of the proposed models scale with dimensions. Further, we empirically validate that each of the design choice help in improving accuracy of learned models and inclusion of heuristic estimators also make the models more robust against data updates.

In summary, we emphasize that regression models have a distinctive advantage in terms of efficiently approximating selectivity for multi-dimensional range predicates. This paper demonstrates that we can design models that deliver both fast and highly accurate estimates. We see this work as a step towards improving the state-of-the-art of selectivity estimation under practical constraints of low overhead query optimization.

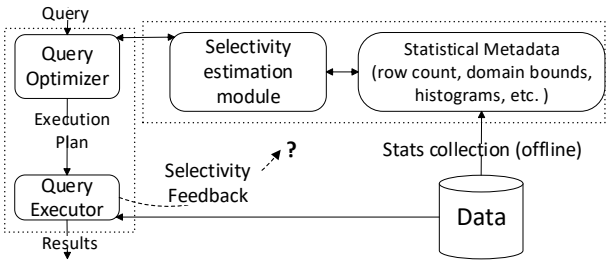


Figure 3: Selectivity estimation in query processing

2. BACKGROUND

When a declarative query is submitted to the database system, it is first optimized to identify a good (low latency) execution plan. The quality of execution plan chosen by the query optimizer hugely depends on the quality of size estimates at intermediate stages of the plan, often termed as *selectivity* estimates. Figure 3 gives an overview of how query processing architecture interacts with selectivity estimation module.

The selectivity estimation techniques usually have an offline phase where they collect statistical information about database tables including row counts, domain bounds and histograms. During query optimization, this information serves as the source for selectivity estimation techniques. For example, consider a query with conjunction of multiple simple predicates ($\langle attribute \rangle$ $\langle operator \rangle$ $\langle constant \rangle$) on different attributes of a database table. Most database systems first compute selectivity fraction[1] for each simple predicate using a histogram on the corresponding attribute. Then, combined selectivity of conjunction is computed using an assumption regarding distribution of values across different attributes. Below, we list two such assumptions:

1. *Attribute Value Independence (AVI)*: It assumes that values for different attributes were chosen independent of each other. Under this assumption, the combined selectivity fraction for predicates on d attributes is calculated as $\prod_{k=1}^{d}(s_k)$, where s_k is the selectivity fraction of predicate on k^{th} attribute.
2. *Exponential BackOff*: When attributes have correlated values, AVI assumption could cause significant underestimations. Microsoft SQL Server introduced an alternative assumption, termed as Exponential BackOff [3], where combined selectivity fraction is calculated using only 4 most selective predicates with diminishing impact. That is, combined selectivity is given by $s_{(1)} \times s_{(2)}^{\frac{1}{2}} \times s_{(3)}^{\frac{1}{4}} \times s_{(4)}^{\frac{1}{8}}$, where $s_{(k)}$ represents k^{th} most selective fraction across all predicates.

Opportunity. The selectivities estimated using limited information available to estimation techniques can be hugely erroneous [27]. For instance, multi-dimensional range predicates on single table may suffer from estimation errors due to attribute correlations. While modern database systems have support for limited multi-column statistics [7], the information is not sufficient to capture the correlation in fine granularity across numeric attributes. While past research literature [14] has already noted that actual selectivities for predicates can be monitored with low overhead during query execution, and can be used to improve future estimates [15, 10, 40] – current systems do not fully exploit this opportunity. Powerful regression methods are excellent candidates that can leverage the feedback information to learn good quality selectivity estimators.

[1]We use the term *selectivity fraction* to denote the fraction of total rows in the table that satisfy the query predicate(s).

3. PROBLEM DESCRIPTION

Consider a table T with d numerical attributes A_1,A_2,..., A_d. Let the domain for k^{th} attribute be $[min_k, max_k]$. Any conjunctive query q on numerical attributes of T can be represented in the following canonical form: $(lb_1 \leq A_1 \leq ub_1) \wedge \ldots \wedge (lb_d \leq A_d \leq ub_d)$. In this representation, if the query does not contain predicate on some attribute A_k, then it is included as $min_k \leq A_k \leq max_k$. For instance, if there are two attributes A_1 and A_2, each with domain [0,100]. Then, predicate $10 \leq A_1 \leq 20$ would have the following canonical representation: $(10 \leq A_1 \leq 20) \wedge (0 \leq A_2 \leq 100)$. The above definition includes one-sided range predicates as well as point predicates, i.e., $A_k = x$ can be specified as $lb_k = x$ and $ub_k = x$.

We define *actual selectivity* of q as the number of rows in table T that satisfy *all* predicates in the query q, and denote it with $act(q)$. Similarly, we use $est(q)$ to denote the estimated selectivity for query q. We define a labeled query set $S = \{(q_1 : act(q_1)), \ldots, (q_m : act(q_m))\}$, with actual selectivity as the label. As an example, $S_{example} = \{((10 \leq A_1 \leq 20) \wedge (0 \leq A_2 \leq 100) : 500), ((10 \leq A_1 \leq 20) \wedge (40 \leq A_2 \leq 80) : 300), \ldots\}$.

Problem. Given a set S of labeled queries, we wish to learn a regression model M, such that for any conjunctive range query q on T, M produces estimated selectivity $est(q)$ close to the actual selectivity $act(q)$.

Note that the problem definition does not make any assumption about the source and distribution of queries in the set S, but the learned model M is expected to produce accurate estimates for queries that are well represented in the training set S. Also, we highlight that our focus is on selectivity estimation for conjunction of two or more predicates, and we do not intend to replace single attribute histograms in the system metadata.

3.1 Formulation as a regression problem

For each query q_j in labeled query set S, we create a tuple of $2 \times d$ values $\langle lb_1, ub_1, \ldots, lb_d, ub_d \rangle$ that serve as input features for the regression model M, we call them *range features*. The corresponding actual selectivity value, i.e., $act(q_j)$ serves as the source of regression label. As an example, the input features for queries in $S_{example}$ are $\langle 10, 20, 0, 100 \rangle$ and $\langle 10, 20, 40, 80 \rangle$ with regression labels 500 and 300, respectively.

Query Space. To answer a query on a d-dimensional dataset, the regression model M takes as input a point location in the $2d$-dimensional space defined over domain of range features – we call it *query space*. The task for regression methods is to learn a function f over the query space to approximate the actual selectivity function. Observe that,

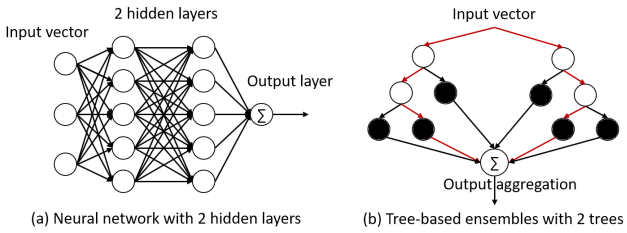


Figure 4: Example of model architecture

only those points in the query space correspond to valid range queries for which $lb_k \leq ub_k, \forall k \in [d]$.

3.2 Evaluation metrics

To evaluate the accuracy of the model M, we use a held-out test query set S_{test} and q-error [32] as accuracy metric. We chose q-error metric since it is relative as well as symmetric [32, 34]. Each query $q_i \in S_{test}$ has a q-error $e_i = \max\left(\frac{est(q_i)}{act(q_i)}, \frac{act(q_i)}{est(q_i)}\right)$. We assume $act(q) \geq 1$ and $est(q) \geq 1$.

To evaluate average accuracy of M over S_{test}, we use *geometric mean* of q-error values, since it is more resilient to outlier errors compared to the arithmetic mean. We also highlight 95^{th} percentile q-error value across all queries.

In addition to the accuracy, we also evaluate the model M on time taken to produce estimated selectivity for queries in S_{test}, and time taken to train the model M using labeled queries in S_{train}.

4. CHOICE OF REGRESSION METHODS

To construct regression model M, we use non-linear regression techniques as the data can have non-linear, complex distributions in general. We consider two types of generic non-linear regression methods: neural networks and tree-based ensembles. Provided with sufficient training data, both methods can learn increasingly more complex regression functions with increase in the number of model parameters and offer flexible accuracy-space tradeoff, as we describe in this section.

4.1 Background

Neural Network (NN). We use the standard feed-forward neural network. The simplest NN model has an *input layer* that consists of *nodes* to feed the vector of feature values as input and a single neuron *output layer* that produces the prediction value. In addition to the neuron in the output layer, the network can have more neurons in the form of l hidden layers. When $l = 0$, neural network is equivalent to a linear regression model. When $l > 1$, the network is referred to as a *deep neural network*. We use neural network with *fully connected* layers, as visualized in Figure 4(a) for $l = 2$. We use *ReLU* (Rectified Linear Unit) activations for neurons in the hidden layers. The neuron in the output layer uses linear activation. The total number of parameters in the model is a function of the number of input features and the number of neurons in each of the hidden layers.

Tree-based ensembles. We consider both random forests and gradient boosted trees. During training, the former uses bagging and the latter uses boosting as the ensemble strategy. Both methods construct multiple binary trees. At the prediction time, each tree independently produces a prediction value using the input features. To do this, each internal node in the tree uses *exactly one* input feature value to decide the next subtree, and each leaf node corresponds to a regression value. The predictions from all the trees are aggregated to produce the final prediction of the ensemble. Random forests use an unweighted average, and gradient boosted trees use a weighted sum to aggregate the predictions. The model sizes for both are determined by the number of trees t and the number of leaves v in each tree. Figure 4(b) shows an example tree-based ensemble model with $t = 2$ trees, each with $v = 3$ leaves where leaf nodes are shown as shaded circles.

4.2 Promise for selectivity estimation

We use regression methods to learn a function over the query space to support multi dimensional range selectivity estimation. In this section, we explain why the chosen regression techniques are promising candidates for multi-dimensional selectivity estimation task.

4.2.1 Neural network

As mentioned earlier, we use ReLU units as the activation functions for hidden neurons. Past work [33, 18] has shown that a neural network with *piecewise linear activations* will divide input feature space (i.e. query space) into local regions, where each region is a convex polytope. Within each local region, the regression function expressed by the neural network is a *linear function* of input features – such functions are called *local linear functions*. The maximal number of local regions is bounded by 2^n, where n is the total number of neurons in the hidden layers. Increase in the number of hidden layers and their neurons leads to increase in the number of possible regions [33], which can help more closely approximate the actual selectivity function.

4.2.2 Tree-based ensembles

Each leaf node in the ensemble of trees corresponds to a region in the query space defined by conjunction of the ranges on input features from the internal nodes along the path from root, e.g.,

$$(c_1 \leq lb_1 < c_2) \wedge (c_3 < ub_1 \leq c_4) \wedge (c_5 \leq ub_2 \leq c_6)$$

Hence, each learned tree partitions the query space using hyper-rectangular regions corresponding to its leaf nodes. Further, different learned trees in the ensemble partition the query space in different ways, where the regions for different trees can overlap with each other. The maximal number of unique predictions is bounded by v^t, where v is the number of leaves in each tree, and t is the number of trees. By increasing the number of leaves, we allow each learned tree to use more regions to partition the query space. With increase in the number of trees, the ensemble can utilize more ways of partitioning the query space. Overall, both can result in finer granularity approximation of the actual selectivity function.

4.2.3 Comparison with multi-d histograms

Multi-dimensional histograms can be viewed as very special regression methods: they store the exact selectivity for a number of 'anchor' queries, corresponding to each histogram

bucket. For example, a bucket for 2d histogram given by $10 \leq A_1 \leq 20$ and $40 \leq A_2 \leq 80$, corresponds to a point location $< 10, 20, 40, 80 >$ in the query space. For any of the 'anchor' queries, multi-d histograms deliver accurate selectivity estimate. For most other queries, the estimate produced by multi-d histograms relies on uniform distribution assumption. While having more such anchor queries can reduce the dependence on uniform assumption to increase accuracy of histograms, their memory and estimation time overhead typically increase linearly with the number of anchor queries.

Our proposed regression methods, on the other hand, process a large number of queries during training, but do not remember any individual query. They use the training queries to effectively learn (1) how to partition the space of all possible queries into regions, and (2) a simple-form selectivity function for all possible queries inside each region. Their unique ability lies in representing a large number of regions using small memory footprint and efficient computation of selectivity estimates, e.g., selectivity estimation with tree base ensembles require traversing small number of binary search trees.

5. MODEL DESIGN CHOICES

While larger size models can achieve finer granularity approximation, they also lead to increased estimation time. This section discusses our model design choices to create compact models that can still capture complex functions.

5.1 Log-transformed labels

We generate training labels by applying log-transform (we use base 2) to selectivity value $act(q_i)$. At estimation time, we apply inverse-transformation on model prediction to get the final estimation, i.e., $est(q_i) = 2^p$, where p is the model prediction. While the transformation is very simple, it enables use of generic regression techniques for problem domains such as selectivity estimation, without changing the implementation details of the technique – this is because selectivity variation across different queries can be huge, and relative metric is more relevant, as explained next.

Increased focus on relative error. We typically expect the regression models to achieve low mean-squared-error (MSE) across all queries, when they work well. Consider an example of two queries q_1 and q_2, where $act(q_1) = 20000$ and $act(q_2)$=100. If we use selectivity values $act(q_i)$ directly as training labels, then the model is more likely to predict $est(q_1) = 19500$, $est(q_2) = 600$, than $est(q_1) = 19100$, $est(q_2) = 200$. Hence, it tends to deliver low relative error for q_1 and high relative error for q_2. Using log transformed labels, i.e. $\log act(q_i)$ pushes the algorithm to reduce the difference between $\log act(q_i)$ and $\log est(q_i)$, i.e.,

$$|\log act(q_i) - \log est(q_i)| = \left|\log \frac{act(q_i)}{est(q_i)}\right| = |\log(e_i)|$$

which should reduce q-error (in general, relative error).

Achieving low MSE in the log-transformed space means low value of

$$\frac{1}{|S_{test}|}\sum_{i=1}^{|S_{test}|}[\log act(q_i) - \log est(q_i)]^2 = \frac{1}{|S_{test}|}\sum_{i=1}^{|S_{test}|}\log^2 e_i$$

Note that, minimizing the average of $log(e_i)$ is equivalent to minimizing the geometric mean of q-error; and minimizing $\max(e_i)$ is equivalent to minimizing the worst case q-error. Optimizing for only one of them might result in undesirable values for another. Minimizing the mean squared $log(e_i)$ accommodates both goals because the square assigns higher weight to larger q-errors during the average.

Implication for NN and tree-based ensembles. We believe that log-transform allows both models to capture abrupt selectivity variation in the query space with fewer parameters. This is because each technique now generates predictions through a *log-linear* function, as explained next.

Neural network As mentioned in Section 4.2, a neural network encodes different linear functions for different local regions. Without doing log transformation, each learned local linear function has the form

$$est(q) = a_0 + a_1 lb_1 + \cdots + a_{2d} ub_d \quad (1)$$

in its local region. By log-transforming the selectivity labels, we allow the NN to learn local *log-linear* functions. Specifically, in a local region, we now have

$$\log est(q) = a_0 + a_1 lb_1 + \cdots + a_{2d} ub_d$$

Equivalently, estimates for each local region are given by,

$$est(q) = 2^{a_0} \times (2^{a_1})^{lb_1} \times \cdots \times (2^{a_{2d}})^{ub_d} \quad (2)$$

In the linear model Eq. (1), each feature value contributes to the selectivity estimate in an additive form. In the log-linear model Eq. (2), each feature value contributes as a multiplicative factor. As a result, even a simple-form function in each local region can accommodate larger selectivity variation.

Tree-based ensembles Given an input feature vector, each learned tree produces a regression value using a subset of features. The ensemble produces final prediction by using a weighted sum of these values. However, when the summed values corresponding to different trees are of very different magnitudes, the values of large magnitude could heavily dominate the final prediction, making it insensitive to the values of small magnitude. That means, although different trees partition the query space into a variety of regions and allow for up to v^t unique predictions, many predictions are merely determined by a small number of trees and have nearly identical value. The log transformation of selectivity labels effectively aggregates the predictions from all the trees with a log-linear model, such that the aggregated value is not heavily dominated by a few predictions in the ensemble. This leads to a potentially more efficient use of the model capacity, because each individual tree can actively influence the final prediction for regions created by various combinations of leaf nodes.

5.2 Correlation based estimators as features

Selectivity is a function of both query and underlying data distribution. As per the formulation in Section 3.1, range features encode information about the query, i.e., query's position and spread in the data domain space. Regression models with only range features learn information about the data distribution indirectly via the actual selectivity labels.

While both neural networks and tree-based models are capable of learning complex interactions between the basic input features, using histograms and domain knowledge to manually engineer extra features can help improve selectivity estimation accuracy without increasing model size if the features are highly informative. On the other hand, since we impose time and size constraint to the regression models, the additional features should satisfy the following basic criteria.

1. **Efficiency**: Efficient derivation of feature values is important since it causes additional overhead to estimation time for an ad hoc query (computing range features is already efficient).
2. **Relevance**: We require the added features to be highly relevant, i.e., strongly correlated with the actual selectivity. Inclusion of irrelevant features may hurt as fewer model parameters would be available to capture the interaction among relevant features, not to mention that it may cause unnecessary computations.

5.2.1 Heuristic estimators as features

The domain experts have designed simple heuristic estimators, e.g., AVI and EBO (described in Section 2) that use information from single attribute histograms to produce selectivity estimates for conjunction of predicates. In the same spirit as AVI, another estimator has been considered in the past [1], which returns the combined selectivity as the minimum selectivity across individual predicates – we call it the MinSel estimator. We include the estimates based on all three heuristic estimators (AVI, EBO and MinSel) as extra input features to our regression models.

All of these estimators have a common first step, i.e., scanning the relevant histograms, followed by estimator specific simple computations. The efficiency is not a concern since single attribute histograms are typically independent of data size. In our experiments, selectivity computation for each attribute took only 20-30 μsec. We justify relevance of these features in the remainder of this section.

5.2.2 Rationale: Varying degree of correlation

For a given conjunction of predicates, the combined actual selectivity depends on the degree of correlation among the individual predicates induced by the underlying data distribution. The above set of estimators can capture various scenarios with different degrees of correlation, as explained next. An estimator based on AVI assumption would produce good estimates if the predicates have no correlation between them. MinSel estimator represents the other extreme compared to AVI, and produces good estimates when all predicates are satisfied by the same set of data rows, i.e., full correlation. The EBO estimator is expected to capture some intermediate scenarios between complete independence and full correlation.

We observe that although each estimator is not accurate for all the queries, it may be accurate for a subset of queries and the model can learn the appropriate mapping from the training data. We include these correlation based features (referred to as CE features) to demonstrate the impact of such features that utilize information in single attribute histograms and we understand that including other estimators may also be helpful.

We empirically validated the relevance of these features for different data distributions. For each estimator, we found specific real-world datasets where it is significantly better than the other two. In Table 1, we show the percentage of queries with q-error ≤ 2 for different estimators across four datasets. The first three workloads have 2-dimensional predicates on 3 different datasets ($dataset_1$, $dataset_2$ and $dataset_3$). We found that each estimator wins for one of the 2D datasets by a margin of at least 10%.

Table 1: Percentage of queries with q-error ≤ 2 for estimators and their oracular combinations on different datasets

Estimators	2D Datasets #1	#2	#3	4D #4
AVI	**94**	61	32	34
MinSel	49	53	**86**	12
EBO	74	**72**	63	37
Oracle combinations				
(AVI,MinSel)	95	72	88	42
(AVI,EBO)	98	80	64	45
(MinSel,EBO)	74	73	89	40
(AVI,MinSel,EBO)	**98**	**81**	**90**	**50**

Our choice of these three estimators as features is further motivated by the observation that the CE features complement each other. We demonstrate their complementary behavior using *oracle combinations* of individual estimators in Table 1. Here, (AVI,EBO) corresponds to a hypothetical oracle that knows the better estimator between AVI and EBO to use for each individual query, and so on. The oracle using all the three estimators has the best performance for all the datasets. For example, it improves the percentage of queries with q-error ≤ 2 from 72% to 81% on $dataset_2$.

We emphasize using the 4D dataset in Table 1 that, while one of these estimators works reasonably well for 2D queries and they also complement each other well, their accuracy for higher dimensional predicates is much worse than 2D queries. For the 4D workload, the percentage of queries with q-error ≤ 2 lies in the range 12% to 50% for all three estimators as well as the hypothetical oracles.

To summarize, choosing one of these estimators irrespective of data distribution clearly exposes database systems to the risk of large estimation errors for majority of queries. However, using such estimators as input features to learned models provides a principled way to exploit their benefits and use model parameters to further improve accuracy.

In addition to improvement in accuracy, we empirically found that use of CE features also improve the robustness of the models against updates to the underlying datasets. The intuition is that 1D histograms can be quickly updated to reflect the data distribution changes and model that uses CE features can use this updated information to improve estimates with respect to the new actual selectivity, the details of the experiment can be found in Section 7.4.

6. INTEGRATION WITH EXISTING DBMS

Regression models for multi-dimensional range selectivity estimation can be integrated into the existing system architecture as shown in Figure 5. At a high level, the integration is conceptually similar to how it would be done for any generic multi-dimensional histograms, as both support estimation of conjunction of range predicates. Below we discuss how these models are utilized by selectivity estimation module, followed by details on training of models.

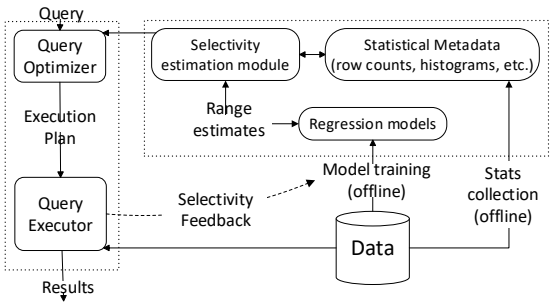


Figure 5: Deployment of regression models

6.1 Estimation with regression models

Consider a model M_1 trained for table T_1 on attributes A_1, A_2, A_3, A_4 and another model M_2 trained for T_2 on attributes B_5, B_6, B_7. Model M_1 can be used to estimate selectivity of conjunctive range predicates over any subset of the attributes A_1 to A_4, by representing the predicate in canonical form defined in Section 3. For example, M_1 can produce estimate for $A_1 \wedge A_2$ or $A_2 \wedge A_3 \wedge A_4$. Similar argument holds for M_2 regarding predicates on T_2. Given that pushing down base-predicates as filter operators on tables (before any join-operator) is a standard practice, models M_1 and M_2 can serve any query that involve table T_1, T_2 or both. If the query has additional predicates on same table, e.g. string predicates or predicates with IN clause, existing methods in database systems or techniques from past work [30] can be used to combine partial information from different sources. Overall, regression models help in fixing estimation errors due to correlations between range predicates at base-level of query plan trees where access path decisions are made. Note that, fixing errors at the base of a plan tree would reduce the error propagated up the tree.

We have empirically evaluated models for up to moderately large (ten) number of attributes. Table that have much larger number of numeric attributes, say 100, pose additional challenges for any technique including regression models. It is unclear whether it is better to create a single model for all attributes or partition them to be handled by separate independent models [23]. It an interesting research problem beyond the scope of this paper.

6.2 Training regression models

Training a regression model requires a set of queries labeled with actual selectivities, and optionally CE feature values. Feedback from past query executions is a natural source to collect such training data, at no explicit overhead. In this respect, regression models resemble query-driven techniques proposed in the past [10, 12, 39, 22]. For any technique that takes only labeled queries as input, we can also think of a scenario where we generate training examples to bootstrap the technique before receiving any external query. Since collecting actual selectivities involves executing a large set of queries over the data, it is quite resource consuming and takes up to (# rows × # queries) operations since queries may have arbitrary overlap with each other. There are several ways to reduce the latency. First, we can use parallel resources, as the training process is offline. Second, we can build an R-tree index on the query set, to reduce the number of queries to be checked for containment per data row. Finally, we can also approximate actual selectivity labels by executing queries over a large enough sample of data and reduce latency by decreasing #rows per query. We empirically study the resources needed for model training under this trade-off in Section 7.4.

Impact of data updates. Models may need retraining whenever there is a significant shift in the query workload distribution or underlying data distribution, compared to initial training. Such cases can be triggered by a bulk data update, similar to the triggers in existing engines [6] or a large fraction of served queries resulting in bad selectivity estimates compared to selectivities found after executions.

Once triggered, the new actual selectivity labels for existing training queries can be computed efficiently if the system already supports a delta store for all updated rows in the data, which reduce # rows to be processed. Otherwise, computing actual selectivities is identical to bootstrapping and similar ideas can be used to reduce overhead. Interestingly, we found that models that use CE features do not necessarily need expensive retraining and can work reasonably well with updated 1D histograms. We study the trade-offs available in case of database updates in Section 7.4.

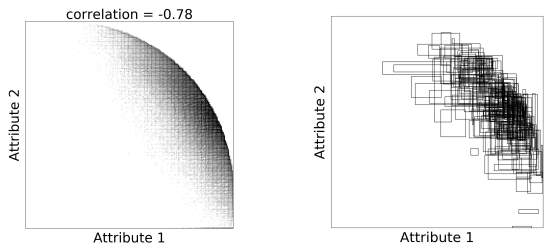
7. EXPERIMENTS

The experimental evaluation is divided under three main categories. First, we evaluate performance of our best models in comparison with existing techniques with varying memory footprint, datasets and predicate dimensionality. Second, we present the decisions that result in best models for a given memory budget, i.e, impact of design choices, hyper-parameters and training size. Finally, we discuss the overhead associated with model training, possible ways to reduce the overhead and impact of data updates on accuracy of models. Before going into details of evaluation, we describe the experimental setup.

7.1 Experimental Setup

Datasets. We used 4 real-world datasets from different domains to evaluate the estimation quality on a variety of realistic data distributions.

1. Forest [8]: The original forest cover type dataset has 581012 rows and 54 attributes. We use first 10 numeric attributes as in [21, 22] (other attributes are binary). An example of its non-linearly correlated attribute-pairs is shown in Figure 6(a).
2. Power [8]: It contains measurements of electric power consumption in a household at one-minute sampling rate over 4 years. We used the 7 numeric attributes after the first two attributes (date and time) with over 2 million rows.
3. Weather: This is a dataset constructed by authors of [34] using data sourced from *daily global historical climate network*. It contains daily observations of climate records in 3.4 million rows across 7 attributes.
4. Higgs [8]: This is a scientific dataset that has 11 million rows with features regarding kinematic properties measured by particle detectors in the accelerator. We use last 7 high-level features that physicists derive to correctly classify the particles.



(a) Data distribution (b) Data centric queries

Figure 6: Example of data-centric query distribution for a correlated attribute pair

Workloads. Each dataset is treated as a table. We use queries with range predicates on two or more attributes. We call a query d-dimensional if it has non-trivial range bounds for d attributes. For a given total number of queries, we construct query workload in the following way: For a D dimensional dataset, we generate queries with dimensions varying from 2 to D for all possible subsets of attributes, with equal number of queries to represent each subset. For a chosen set of d attributes $(A_{k_1}, A_{k_2}, ..., A_{k_d})$ of a dataset, we create queries in the manner described next. The query centers follow one of the two distributions in equal proportions: uniform random over the data domain, or uniform random over the data tuples. The first distribution is named random-centric and the second distribution is named data-centric distribution, as visualized in Figure 6(b). For each dimension of a query, we use uniformly distributed widths for random-centric queries and exponentially distributed widths for data-centric queries, over the domain of that dimension. We validated that the constructed query sets contain queries all over the data domain space, with huge variation in volumes as well as actual selectivity values. Unless stated otherwise, we used query workload with 20,000 examples, with 80% queries as training set and remaining 20% as test set.

List of comparative techniques. We used the following techniques for comparison.

1. **AVI**, traditional estimator for most database systems.

2. **STHoles** [12] (abbv. as sth), state-of-the-art query driven technique for multi-d histograms.

3. **Samples**, i.e, uniform random samples, given the same size constraint as other techniques. We use a sample with 1000 rows (abbv. as 1k_sample) when comparing with fixed small size (16KB) models.

4. **KDE[22]**, (abbv. as *sample_kde*) is a state-of-the-art kernel density estimator that can use feedback queries to improve the estimation quality for any given sample. We used the code made available by the authors [2] with *SquaredQ* loss function and *Batch* variant for bandwidth optimization.

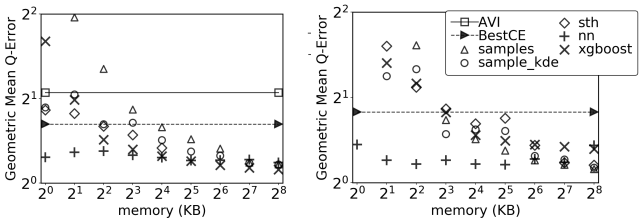
5. **BestCE**, refers to an *oracle* technique that can magically choose the most accurate selectivity estimate among AVI, EBO, and MinSel for any given query. When BestCE does not improve upon AVI, it highlights the difficulty of estimation task. [2]

6. **10k_sample**, we also use uniform random sample with 10k rows for accuracy comparison. Observe that, it is larger than typical recommendation [7] and has longer estimation time ($> 10\times$ compared to AVI).

The first four are existing techniques, and the latter two impractical methods serve as accuracy landmarks.

Model variants. We experimented with three different kinds of regression techniques, i.e., neural network implemented with Keras [17]), random forest [36], and gradient boosted trees, XGBoost [16] and LightGBM [24]. We found that XGBoost has equal or better accuracy and estimation time than both LightGBM and random forests for a given model size, so we only report XGBoost (denoted as xgboost) and neural network (denoted as nn).

For each regression model, there are *four variants* due to our design choices: Log-transform and CE features. We propose the variant that uses both choices (denoted as Log+CE) as the best variant. Hence, only Log+CE variant is used in most experiments except for Section 7.3.1, where we empirically justify our choice. For each model size budget, we consider a space of models, i.e., nn with different numbers of hidden layers and xgboost with different numbers of trees, as detailed in Section 7.3.2. We report the performance for best choice model for each memory budget, except in Section 7.3.2, where we present the typical choices that we empirically observed for the best models. Note that, range features are normalized to domain [0,1000]; log-transformation of labels use log-base 2; and CE features go through the same transformation as the labels.



(a) Forest$_{2D}$ (b) Power$_{2D}$

Figure 7: Accuracy for correlated attribute-pairs

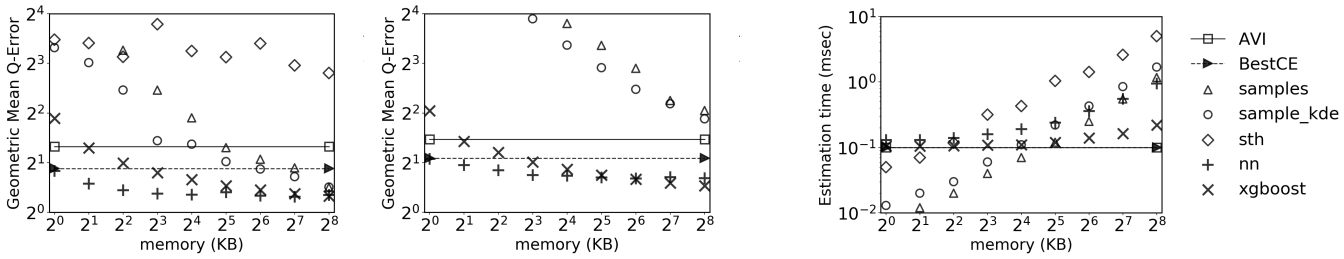
7.2 Accuracy and estimation time evaluation

We first study the impact of memory on accuracy and estimation time of different techniques. Note that, AVI and BestCE are baselines that use 1D histograms with at most 200 buckets each. Then we compare our models constrained on memory and estimation time with other techniques across different datasets and predicates of different dimensionality.

7.2.1 Impact of memory budget

In this section we plot, on a log-log scale, the accuracy measured by geometric mean of q-error (later abbv. as gmq),

[2]By definition, the accuracy of BestCE is better than each of AVI, EBO, and MinSel – hence we do not report explicit comparison with EBO and MinSel.



(a) Accuracy for Power$_{4D}$ (b) Accuracy for Power$_{7D}$ (c) Estimation Time for Power$_{4D}$

Figure 8: Impact of model size on accuracy and estimation time

for all techniques over a range of memory budgets. We also report estimation time over the same memory range. Together, they show that larger models typically give better accuracy at the expense of increased estimation time.

2D correlated datasets. In Figure 7, we use (as datasets) two targeted attribute pairs with different types of correlation: (a) non-linearly correlated attribute-pair, visualized in Figure 6(a), from Forest dataset, and (b) linearly correlated attribute-pair from Power dataset. Below are the highlights from this experiment,

- BestCE is significantly more accurate compared to AVI when attributes are correlated.
- All techniques including our models gain accuracy very quickly with memory to outperform even BestCE.
- NN models are highly accurate for model sizes as small as 1 KB, i.e., 128 parameters. XGBoost models typically need more space compared to NN models.

Accuracy for higher dimensional workloads. In Figure 8, we show the performance of different techniques for a representative higher dimensional dataset (Power). We highlight here that, our workload includes queries on all possible subset of attributes. Figure 8(a) and (b) show accuracy for 4-attribute subset and the full 7-attribute version of Power dataset, respectively. Figure 8(c) show the estimation time (in millisecond) for the 4D subset.

BestCE continues to be more accurate compared to AVI even for higher dimensional workloads. For sampling, more memory allows more sample tuples that leads to improved accuracy. As expected, KDE technique consistently outperforms sampling and performs quite similar to AVI estimator for small sample sizes for 2D as well as 4D workloads. But when evaluated on 7D version of the dataset (Figure 8(b)), the accuracy gain is not sufficient to outperform the AVI estimator in terms of *gmq*. While it is already known that improvements due to KDE technique reduce with increase in dimensions [22], we suspect that the mix of 2D to 7D predicates in our workloads further increases the difficulty of bandwidth optimization. We do not use parallel-version of KDE [22] that could efficiently process larger samples.

STHoles, in marked contrast to the 2D dataset, could not deliver reasonable accuracy even with 256 KB memory, while its estimation time increased rapidly. Note here that STHoles does not directly optimize for the relative metric. For 7-dimensional version of Power dataset shown in Figure 8(b), the accuracy degrades further (out of plot range). We believe that one reason for huge degradation in accuracy is highly intersecting queries spreading all over the large 7-dimensional domain space.

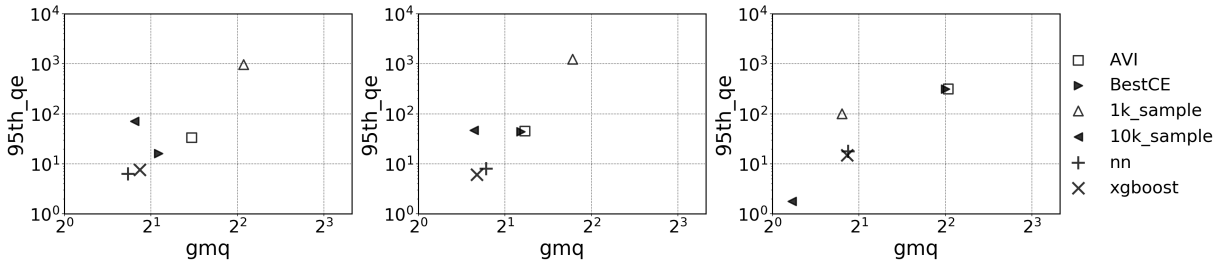
Finally, regression models with memory footprint as small as 16KB are sufficient to outperform the stronger baseline BestCE even for higher dimensional workloads.

Estimation time. With regard to estimation time, 16KB regression models are within 2× factor compared to AVI that takes $\approx 100\mu$sec. Observe that the estimation time increases very slowly for XGBoost models. The time for models corresponds to the variant Log+CE and includes the time to compute the heuristic estimators required for input features. Sampling needs much longer estimation time ($\approx 10\times$), to deliver accuracy values similar to 16KB models. These estimation times are computed using a single-threaded implementation of only the prediction module of each technique to avoid platform bias. Specifically, estimation routine for our NN models requires $(l+1)$ vector to matrix multiplications and additions, and XGBoost models require binary tree traversal for each learned tree. We have reported times for NNs with 3 layers and XGBoost trees with moderate depth (16 leaves). Also, estimation time for models does not change significantly with small change in number of input features, hence the behavior shown in Figure 8(c) is representative for wider datasets as well.

7.2.2 Accuracy across different datasets

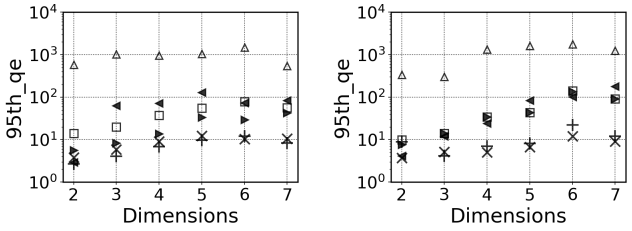
Given their fast estimation time, we use models with small memory footprint (16 KB) in all further experiments. We compare them against other techniques that have short estimation time (AVI, and 1k_sample), as well as the accuracy landmarks, BestCE and 10k_sample. KDE and STHoles are not included as they are not expected to deliver competitive accuracy values under the estimation time constraint. In addition to geometric mean q-error (in the range [1,10]), we report 95^{th} percentile q-error (range [1,10^4]) across 3 datasets in Figure 9 (Higgs dataset is used later in Section 7.2.3).

Observe that lightweight regression models consistently outperform all other techniques in both metrics. The only dataset for which our models do not reach the 10k_sample landmark is Forest dataset, because the dataset itself is small ($\approx$ 500k rows). The accuracy for AVI estimator and 1k_sample are almost never satisfactory; and BestCE improves over AVI only for Power dataset. In absolute terms, models have *gmq* values close to 2 and 95^{th} percentile close to 10.



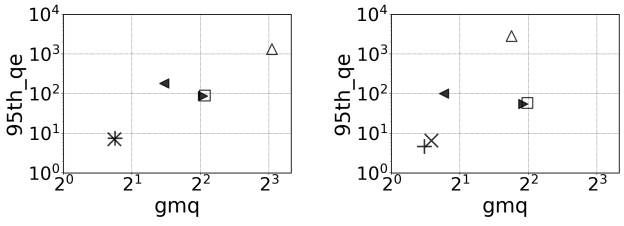
(a) Power (b) Weather (c) Forest

Figure 9: Accuracy comparison for various datasets (models take 16 KB each)



(a) Power (b) Weather

Figure 10: Impact of increase in number of dimensions (legend shared with Figure 9)



(a) Weather (b) Higgs

Figure 11: Accuracy for only 7D predicates on large datasets (legend shared with Figure 9)

7.2.3 *Scaling with the number of dimensions*

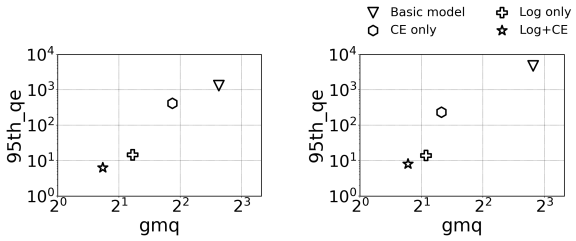
Next, we evaluate the impact of dimensionality on various techniques. For this purpose, we first dig deeper into the accuracy of the (16KB) models evaluated in Section 7.2.2 by breaking down accuracy with regard to dimensionality of the predicates. In Figure 10, for Power and Weather datasets, we report the 95^{th} percentile q-error values on y-axis (log-scaled), with the number of dimensions on x-axis. For a given dimension, say d, we evaluate queries on all possible d-dimensional predicates on various attribute subsets.

We find that 1k_sample is the least accurate technique. Queries with more dimensions tend to be more selective, which leads to drop in accuracy of sampling techniques with small samples. BestCE, AVI and 10k_sample, all of them suffer from 95^{th} percentile q-error close to 100 for high dimensional queries. We show that the accuracy of our models is significantly more robust with the increase in dimensions, q-error keeping below 10 at 95^{th} percentile in most cases.

Finally, we present an experiment targeted specifically for high dimensional predicates on large size datasets (Weather and Higgs). In Figure 11, we show that the accuracy of models is significantly better compared to even 10k_sample when we consider only large size datasets. Observe that our models perform significantly better than AVI and sampling techniques in terms of *gmq* values as well, showing that our models are particularly better at handling high dimensional predicates.

7.3 Best model selection

In this section, we discuss the choices that help us select best regression model for a given memory budget (we use 16 KB budget).



(a) Power (b) Weather

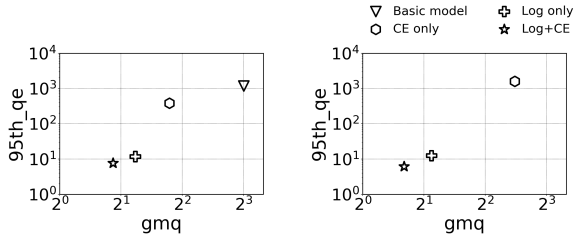
Figure 12: Design impact on NN models

7.3.1 *Impact of design choices*

First, we evaluate the impact of proposed design choices on the accuracy of models. We present the accuracy for all model variants for NN models in Figure 12 and for XGBoost in Figure 13. Observe that the accuracy improves, with respect to Basic Model, with each individual design choice. We find that individually *log-transform* leads to improvement of larger magnitude compared to *CE features*. We emphasize that CE features play a crucial role of delivering "last mile improvements". Overall, the design choices help both NN and XGBoost models. The cost of improvements due to CE features is the extra memory requirement in terms of 1D histograms. We do not consider memory for histograms as an overhead for our models since histograms are already present in systems and may be useful for other purposes.

7.3.2 *Model architecture space*

As mentioned earlier, we explored different models for a given memory budget and reported only the best models for



(a) Power (b) Weather

Figure 13: Design impact on XGBoost models

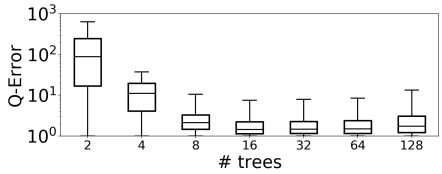


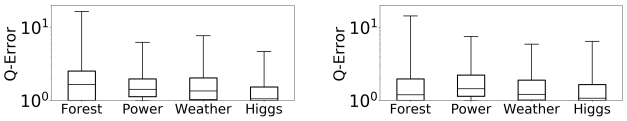
Figure 14: Accuracy impact of #trees: XGBoost (16 KB)

each budget. In this section, we describe how we explored model architecture space for a given model size.

For neural networks, given a model size constraint, we construct different networks by varying the number of hidden layers l from 1 to 4. For each choice of l, we create a single model such that the number of neurons in all hidden layers are as close as possible while satisfying $n_i \geq n_j$ for each pair of adjacent layers ($j = i + 1$). Each network is optimized for 500 epochs using *Adam* algorithm, with batch size 32 and *mean squared error (MSE)* as the loss function.

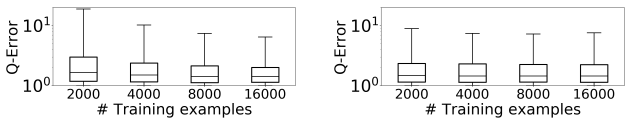
For tree-based ensembles, for a given model size constraint, we create different models by varying the number of trees as powers of 2, starting at two trees. That is, $t \in \{2, 4, 8, ...\}$. For each choice of t, we then decide the number of leaves as the maximum number among $v \in \{2, 4, 8, ...\}$ that satisfies the model size constraint. We used single-threaded training with default values for other hyperparameters.

We empirically observed that neural networks with $l = 2$ or $l = 3$ hidden layers provide reasonable accuracy across all datasets. For XGBoost models, we found that a small number of deep trees as well as many trees with small depth provide suboptimal accuracy (Figure 14). For a given model size constraint, it is most reasonable to use moderately deep trees (16 leaves each) and increasing the number of trees as memory budget increases, our 16 KB models accommodate 16 learned trees. For 16 KB models with these architecture choices, Figure 15 shows the distribution of q-error values across different datasets using boxplots. Note that the box



(a) NN (b) XGBoost

Figure 15: 3-layer NN and 16-tree XGBoost (16KB)



(a) NN (b) XGBoost

Figure 16: Accuracy gains with # training examples

boundaries represent 25^{th} and 75^{th} with the internal marker as median value. We use the whisker to represent 95^{th} percentile error value. Note that for Higgs dataset, we report the models targeted to learn high-dimensional queries.

7.3.3 *Trade-off with training set size*

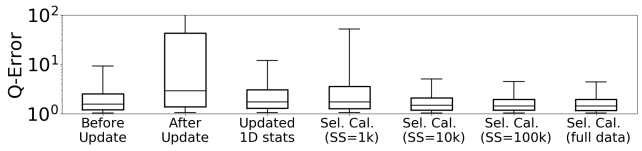
In this section, we analyze the impact of increased training data on accuracy of regression models (shown in Figure 16). We report these numbers for 16 KB models with 3 layers for NN and 16 trees for XGBoost on Power dataset. The accuracy generally improves with larger training data for both model types. However, NN models typically require more training data while XGBoost models work reasonably well even with small training.

7.4 Overhead associated with models

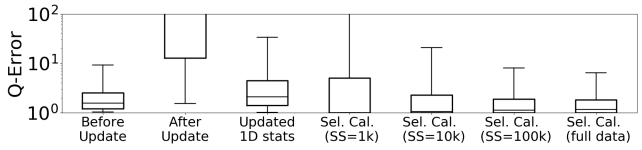
In this section, we discuss the overhead associated with training and maintaining regression models. The entire pipeline to train the proposed regression models consists of three major steps. The first step is construction of the training queries labeled with actual selectivities. The second step is computation of range and CE features, that in turn require constructing/updating 1D histograms. And the final step is using the training data to learn the model parameters. As mentioned in Section 6.2, collecting actual selectivity labels is the most resource consuming step as it is proportional to data size itself, unless the labels are available from query feedback. Also, the steps are identical for initial model training as well as retraining after data updates, unless the system supports a delta store for the updated rows. We first discuss the trade-offs associated with model training in case of updates. Then, we compare the end-to-end training overhead for NN and XGBoost models.

Handling data updates. We use Power dataset for this experiment and present results with XGBoost as the regression model. Note that, the original dataset has 2M rows and a variety of pair-wise correlations. We performed two experiments (visualized in Figure 17) where: (a) we appended 20% extra rows in sparse regions of the original dataset; (b) we updated all tuples of the dataset resulting in huge change in the data distribution (no significant change in pair-wise correlations).

First, we focus on the leftmost two box-plots in Figure 17(a). We observe that that if we continue to use the models trained using selectivity labels and CE features from the original dataset, the accuracy of our models degrades significantly even with 20% appended data rows. In Figure 17(b) where 100% rows are updated, the accuracy degradation is much worse. Hence, if the underlying data is significantly updated, the models are not recommended to be used in their original form.



(a) 20% rows appended



(b) 100% rows updated

Figure 17: Handling dataset updates

Interestingly, we found that the accuracy of our models can be regained to a large extent by only updating 1D histograms and using updated CE features as input to the originally trained model, as shown by the third box-plot in each of Figure 17(a) and (b). Observe that, histogram update is comparatively fast and is already supported by existing systems. Thus, use of CE features makes our models more robust against data updates.

Model retraining overhead. To achieve best accuracy with regression models, selectivity labels used during training should reflect the actual data distribution. In this experiment, we empirically show that use of data sample for selectivity calculations provides an interesting trade-off between overhead of calculating selectivity labels and accuracy. We used 1k, 10k, 100k sample in addition to full data (2M rows) and the resulting q-error distributions are shown in last four box-plots of Figure 17. We found that use of 100k sample (5% of the original dataset) provides reasonably good accuracy with 20× less data to process.

Finally, we found that XGBoost takes only few seconds for the last step of learning model parameters compared to several minutes for NN. To measure end-to-end training overhead, we used 2k training examples for XGBoost model and 16k examples for NN with selectivities computed using 100k sample. The entire process took less than a minute to learn XGBoost model, 10× faster compared to NN model.

8. RELATED WORK

Selectivity estimation in the context of query optimization has been an active area of research for multiple decades – we refer to [19] for an extensive survey. Here, we provide a brief review of studies that are relevant to the goal of selectivity estimation for conjunctive range predicates.

Multi-dimensional histograms This is arguably the most well studied approach to capture attribute correlations [35, 37]. The idea is to use multi-dimensional buckets that partition the data domain, to approximate the data distribution. This approach faces challenges as dimensions increase because there are potentially many ways to identify buckets [19]. Also, the space requirement increases with the number of dimensions because the domain space increases exponentially and data can be skewed. Later proposals [12, 21, 38] used *overlapping* buckets that allow a given number of buckets to identify more complex distributions, they generally require more resources during bucket identification.

Query-driven histograms Query-driven (self-tuning) methods [10, 12, 39] are better designs in multi-dimensional histogram as they focus only on regions in the subspaces that are being accessed by queries in the current workload. The limitation of such methods is highlighted when workload is spread in large fraction of a high-dimensional space.

Sampling based methods Sampling based methods [42] have a lot of advantages as a data sample can represent any distribution without prior knowledge and support a richer class of predicates beyond range predicates. Their key weaknesses are (a) low accuracy for highly selective predicates and (b) high estimation cost because calculating an estimate requires a full scan over the samples [22]. Kernel density estimators (KDE) [21, 22] can provide significantly improved accuracy for given sample size. But such techniques [22] are designed for futuristic platforms when GPUs are commonplace. Also, they do not avoid scanning a large sample at estimation time, which is slow without parallel resources. There have been recent proposals [34, 25] that combine information from both histograms and samples to handle the issue of highly selective predicates, but they do not meet the practical requirements of small memory footprint and estimation time.

Learned models Use of machine learning techniques for selectivity estimation is not new, prior work explored different models ranging from curve-fitting[15], wavelets [31], to probabilistic graphical models [20, 41]. Infact some early works also use neural network based models for selectivity estimation over small number of attribute [11, 26, 29, 28]. Despite these attempts, fast and accurate selectivity estimation for multi-dimensional range predicates was a blindspot. In this paper, we demonstrate significant progress on this important problem with extensive evaluation on multiple real-world datasets. Most recently, [25] targeted correlations across joins using a custom neural network architecture. While [25] certainly addresses generic version of the selectivity estimation problem, the models in this paper are much more succinct leading to significantly faster estimations. Also, we do not confine the regression techniques to be neural network, and found that tree-based ensembles are much faster to train compared to neural networks.

9. CONCLUSION AND FUTURE WORK

This paper explored the application of standard regression techniques to selectivity estimation of range predicates. We showed that lightweight models can be designed to to deliver fast and accurate estimates for multi-dimensional range predicates. With extensive empirical evaluation over multiple real world datasets, we found that the accuracy of models is significantly better than existing methods in database systems, with reasonably small training effort. We believe that the learned models can add significant value to existing systems, due to desirable properties such as small memory footprint and simple estimate routines.

There are several interesting directions for future work including automatically deciding the attribute subsets for model construction; handling larger class of queries such as join queries with arbitrary filters etc. We see this work as an initial step towards using learning techniques to improve the state of selectivity estimation under practical constraint of small query optimization time.

10. REFERENCES

[1] https://support.microsoft.com/en-us/help/2658214/fix-poor-performance-when-you-run-a-query-that-contains-correlated-and.

[2] https://bitbucket.org/mheimel/feedback-kde/src/default/.

[3] Cardinality estimation for correlated columns in SQL Server 2016. https://blogs.msdn.microsoft.com/sql_server_team/cardinality-estimation-for-correlated-columns-in-sql-server-2016/.

[4] How the planner uses statistics. https://www.postgresql.org/docs/current/row-estimation-examples.html.

[5] Statistics in Microsoft SQL Server 2017. https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-2017.

[6] Statistics in Microsoft SQL Server 2017: When to update statistics. https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-2017#UpdateStatistics.

[7] Understanding optimizer statistics with oracle database 18c. https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-stats-concepts-0218-4403739.pdf.

[8] UCI machine learning repository. https://archive.ics.uci.edu/ml/index.php.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, 2016.

[10] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *SIGMOD*, 1999.

[11] J. Boulos and M. Bretonneux. Selectivity Estimation Using Neural Networks. 1996.

[12] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *SIGMOD*, 2001.

[13] S. Chaudhuri. Query optimizers: Time to rethink the contract? In *SIGMOD*, 2009.

[14] S. Chaudhuri, V. Narasayya, and R. Ramamurthy. A pay-as-you-go framework for query execution feedback. *PVLDB*, 1(1):1141–1152, 2008.

[15] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *SIGMOD*, 1994.

[16] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, 2016.

[17] F. Chollet et al. Keras. https://keras.io, 2015.

[18] L. Chu, X. Hu, J. Hu, L. Wang, and J. Pei. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *KDD*, 2018.

[19] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, 4(1-3), 2012.

[20] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, 2001.

[21] D. Gunopulos, G. Kollios, J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *The VLDB Journal*, 14(2), 2005.

[22] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, 2015.

[23] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, 2017.

[25] 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*, 2019.

[26] M. S. Lakshmi and S. Zhou. Selectivity estimation in extensible databases - a neural network approach. In *VLDB*, 1998.

[27] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal*, 27(5), 2018.

[28] H. Liu, M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte. Cardinality Estimation Using Neural Networks. In *CASCON*, 2015.

[29] H. Lu and R. Setiono. Effective query size estimation using neural networks. *Applied Intelligence*, 16(3), 2002.

[30] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *The VLDB Journal*, 16(1), 2007.

[31] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, 1998.

[32] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB*, 2(1):982–993, 2009.

[33] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

[34] M. Müller, G. Moerkotte, and O. Kolb. Improved selectivity estimation by combining knowledge from sampling and synopses. *PVLDB*, 11(9):1016–1028, 2018.

[35] M. Muralikrishna and D. J. DeWitt. Equi-depth multidimensional histograms. In *SIGMOD*, 1988.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[37] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, 1997.

[38] M. Shekelyan, A. Dignös, and J. Gamper. Digithist: A histogram-based data summary with tight error bounds. *PVLDB*, 10(11):1514–1525, 2017.

[39] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. Isomer: Consistent histogram construction using query feedback. In *ICDE*, 2006.

[40] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. Leo - db2's learning optimizer. In *VLDB*, 2001.

[41] K. Tzoumas, A. Deshpande, and C. S. Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1), 2013.

[42] Y.-L. Wu, D. Agrawal, and A. El Abbadi. Applying the golden rule of sampling for query estimation. In *SIGMOD*, 2001.