Learning Based Approach in the Selectivity Estimation Problem

Author: Liana Darbinyan
BS in Data Science
AUA

Supervisor: Aleksandr Hayrapetyan
Yerevan State University
AUA

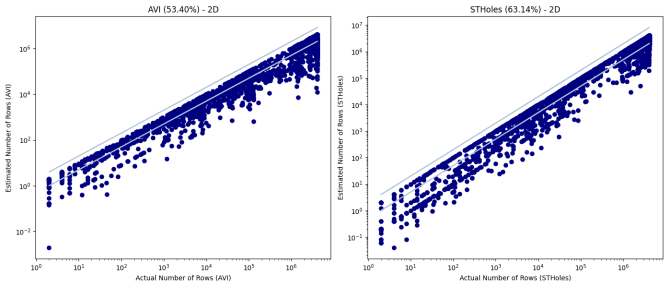
***Abstract*—Coherent query optimization depends on the accuracy of Selectivity Estimation (SE) in generating efficient execution plans. Modern optimization techniques use diverse assumptions, such as predicate independence, when dealing with queries containing multiple predicates. However, in contrast with the advantage of fast estimation time and less memory consumption, they often suffer from large selectivity estimation errors. This capstone thesis presents a comprehensive investigation of selectivity estimation, considering it a regression problem. We explore the application of neural networks and tree-based ensembles to the vital problem of selectivity estimation of multi-dimensional range predicates. Focusing on database system performance, we conducted a deep analysis of several datasets from various fields for collecting and analyzing statistical data, aiming to enhance selectivity estimation within the system. By performing an in-depth statistical evaluation of numerous datasets, we demonstrate that the suggested models produce estimates that are both fast and highly accurate.**

***Index Terms*—Keywords: Relational databases · Query optimizer · Cardinality estimation · Neural Networks · Machine Learning for Selectivity Estimation**

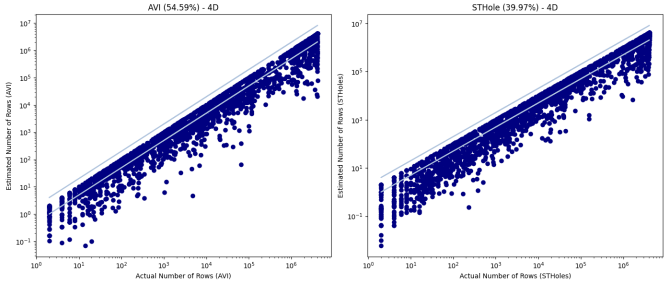
I. Introduction

Selectivity Estimation (SE) is the central component in cost estimation, used for database query optimization that involves estimating the size of query predicate results [1]. This estimation forms the basis for the optimizer to select the most cost-effective execution plan. It gathers attribute statistics like data distribution, influencing planning decisions to achieve smaller intermediate result sizes for more efficient execution. Ideally, an SE technique should provide accurate and fast estimates, use a data structure that has a small memory footprint and be efficient in constructing and maintaining [2]. The requirement of fast estimation follows from the expectation that query optimization time should be short [3], [4] and is an essential practical constraint in the database systems. In cost-based optimization, SE assesses alternative query plans' costs, selecting the most resource-efficient one out of them all. Existing methodologies include heuristic-based assumptions such as Attribute Value Independence (AVI) for calculation of combined selectivity on multidimensional predicates [4], [5], construction of histograms on single-table attributes [4], or integration of lightweight models, strategies for handling inequality joins and deep learning applications in multi-attribute scenarios. These often fail to provide precise estimates, leading to sub-optimal or inefficient execution plans and eased system performance. In the cases of heuristic-based estimation assumptions, having attribute correlation may raise huge errors, resulting in low-quality plans [6], [7]. In such situations building multidimensional histograms might help to some extent, but frequently, significant space is required. Approaches based on sampling could be effectively successful in managing correlations and attribute dependencies, however in the case of queries with low selectivity, the optimizer could be forced to depend on magic constants [8], leading to inaccurate estimations.

Researchers have been actively addressing the multi-dimensional selectivity estimation problem [2], resulting in various approaches such as multi-dimensional histograms [9]–[11], techniques utilizing random samples [12], or using hybrids of histograms and samples [13]. Those techniques enhance accuracy by substantially raising either space or time requirements. This is mainly done because they rely on a greater number of histogram buckets or larger sample sizes



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



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

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

to adequately depict the data distribution in high-dimensional space. Query-driven histograms, also called self-tuning histograms [9], [12], [14], [15], employ more histogram buckets in the subspace of the current query workload. Nevertheless, this focused strategy aims to refine the balance between precision and expense, the precision of such methods might degrade when workload queries have more intersections with one another, spanning across a large portion of multi-dimensional space.

In the scope of our capstone project, we analyzed estimates generated using two types of histogram techniques: (i) AVI assumption, which solely relies on one-dimensional histograms; and (ii) STHoles [9], which employs a query-driven multi-dimensional histogram. We utilized a representative real-world datasets and a set of queries distributed across the entire domain space. In Figure 1, a scatter plot illustrates the actual number of rows satisfying the predicates (s) versus the corresponding estimated values ($\hat{s}$) for both 2D and 4D predicates. It can be noted that queries with small estimation errors, where $\max(\frac{s}{\hat{s}}, \frac{\hat{s}}{s}) < 2$, lie in a diagonal band-shaped area. Queries with larger errors deviate farther from this region. With the AVI approach, small part of the queries exhibit small errors, while large errors are relatively common. In contrast, STHoles provides estimates with higher accuracy for 2D predicates except for some outlying ones. However, for 4D predicates it results in significant errors and larger estimation time.

Beyond the above-mentioned methods, the issue of 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" [1]. Such labeled queries can be obtained as feedback from prior query executions [14], [16], as suggested by self-tuning techniques, or they can be constructed offline in data-driven manner, similar to histogram construction. Regression formulation efforts in past attempts employed neural networks [17]–[21], however, these methods are not designed for fast estimation of multi-dimensional range selectivity.

II. Literature Review

Whenever a declarative query is submitted to the database system, it undergoes an optimization process to identify an efficient (low latency) execution plan. The parsed query is being transformed in the query decomposer, which is given to the query optimizer, where the core of the query execution plan is being generated. The execution plan is chosen by the query optimizer, and its quality heavily relies on the accuracy of size estimates at various intermediate stages, often referred to as *selectivity estimates*. Selectivity estimation is a critical aspect of query optimization as it directly influences the efficiency of query processing. Figure 2 provides an overview of how the query processing architecture interacts with the selectivity estimation module. The selectivity estimation technique commonly has an offline phase, which is the statistical information collection process when we excite the information about database tables with row counts and domain bounds. This information serves as a source for selectivity estimation

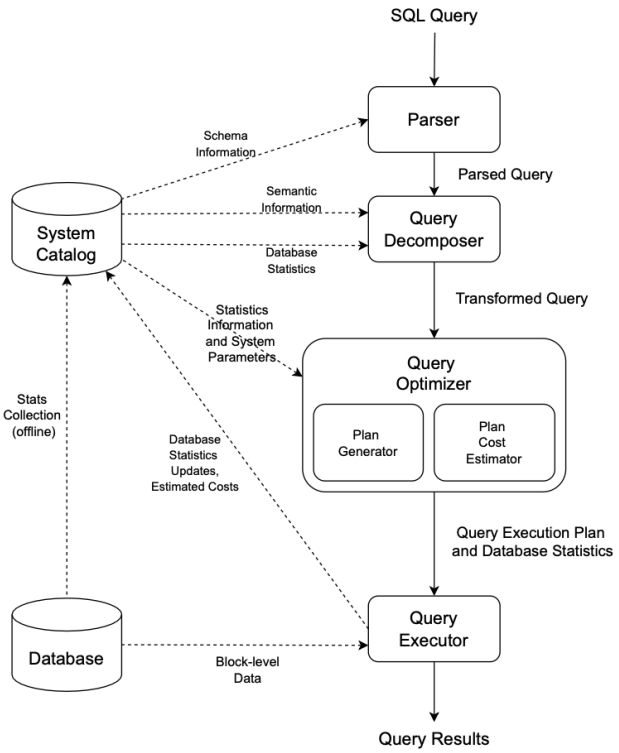


Fig. 2: Selectivity estimation in query processing

techniques in the process of query optimization. An example of a query with conjunction of multiple simple predicates on different attributes of a database table can be represented in a general form: ($\langle$*attribute*$\rangle$ $\langle$*operator*$\rangle$ $\langle$*constant*$\rangle$). In most cases, database systems calculate selectivity for each simple predicate as a first step, choosing fractional approach and constructing a histogram on the corresponding attribute. After that, the combined selectivity of conjunction is being calculated using an assumption of distribution of values for different attributes. Two of the most common assumptions are:

1) *Attribute Value Independence (AVI):* This method relies solely on one-dimensional histograms. It assumes that the selectivity of each attribute is independent of the values of other attributes in the query predicate. AVI assumption leads to the implication of having no correlation between the selectivity of different attributes within each query. While the estimation process is being simplified with this assumption, in the majority of cases it may lead to inaccuracies, especially in high-dimensional spaces where correlations between attributes cannot be monitored and are common. Under this assumption, the combined selectivity fraction for predicates is calculated as

$$\prod_{k=1}^{d}(s_k)$$

where d is the attributes and s_k is the selectivity fraction of the predicate on the k$^{\text{th}}$ attribute. AVI is often com-

pared to other techniques, such as query-driven multi-dimensional histograms, which take into account the correlations between different attributes to provide more accurate selectivity estimates.

2) *STHoles — a "workload aware" histogram:* This method allows bucket nesting to capture data regions with uniform tuple density. STHoles histograms are created without data sets examination, rather by analyzing query results. Buckets used are being allocated based on the most need of the workload; this in its turn, leads to accurate selectivity estimation of the query [9]. Multidimensional histograms are considered to be an effective approach for accurate selectivity estimation in case of multi-attribute queries. This technique is widely used for its possibility of producing good estimates across different kinds of datasets such as synthetic, real-world data and across query workloads. In many cases it even outperforms the best multidimensional histogram techniques which require access to and processing of the full data sets during the histogram construction.

The existing techniques for selectivity estimation highly depend on the information availability, and in cases of limited insights, those techniques may result in inaccurate calculations or huge time and memory requirements [5]. As an instance, due to attribute correlations, multi-dimensional range predictions on a single database can yield to estimation mistakes. Past research literature states that actual selectivity for predicates can be used to improve future estimates - current systems do not fully satisfy this yet. Many powerful regression methods are present for using the query results for learning and acquiring good quality selectivity estimation.

A. Problem Statement

In the scope of this capstone project, we performed improvement of database query optimization, with a specific focus on refining selectivity estimation methods. Aiming to address the issue of inaccurate selectivity prediction for range predicates and queries involving multiple attributes, by delving into innovative approaches. Existing approaches often face limitations in providing precise estimations, resulting in sub-optimal query execution plans that significantly impact the overall performance of database systems. We aim to address this issue by exploring and implementing advanced techniques, including the integration of traditional machine learning models for range predicates, formulation of known strategies to enhance selectivity estimation, and application of deep learning in scenarios with multiple attributes. The overarching goal is to forge more accurate and adaptable selectivity estimation methods, ultimately contributing to the enhancement of database query optimization and, consequently, improving the overall efficiency of database systems across various use cases.

We consider having a table T, with d numerical attributes $A_1, A_2, ..., A_{\mathrm{d}}$. The domain for k^{th} attribute is considered to be $[min_{\mathrm{k}}, max_{\mathrm{k}}]$. Conjunctive query *q* on numerical attributes of T are being represented in the following canonical form: $(lb_1 \leq A_1 \leq ub_1) \wedge ... \wedge (lb_{\mathrm{d}} \leq A_{\mathrm{d}} \leq ub_{\mathrm{d}})$, where *lb* is the lower bound and *ub* is the upper bound of the attribute in the query. In this representation, those bounds for each attribute are defined based on the query that the attribute is involved in, shrinking the domain values by the corresponding limitations. For queries that do not contain predicates on some attribute $\mathrm{A_k}$, the domain values are being used as $[min_{\mathrm{k}} \leq A_{\mathrm{k}} \leq max_{\mathrm{k}}]$. In the instance of having two attributes $A_1 and A_2$, each having domain [0, 100], the predicate $A_1 \leq 10$ would result in the following canonical representation: $(0 \leq A_1 \leq 10) \wedge (0 \leq A_2 \leq 100)$. The actual selectivity of *q* is defined as the number of rows in table T satisfying all predicates in the query *q*, denoting it as *act(q)*. The same way, the estimated selectivity for query *q* is denoted as *est(q)*. This gives us the opportunity to define a labeled query set

$$S = \{(q_1 : act(q_1)), (q_2 : act(q_2)), ..., (q_{\mathrm{m}} : act(q_{\mathrm{m}}))\}$$

with the actual selectivity as the label. Example of this query set can be

$$\begin{aligned} S_{\text{example}} = \{&((0 \leq A_1 \leq 10) \wedge (0 \leq A_2 \leq 100)) : 10000, \\ &((15 \leq A_1 \leq 100) \wedge (2 \leq A_3 \leq 17)) : 44650, \ldots, \\ &((0 \leq A_2 \leq 36) \wedge (16 \leq A_6 \leq 20)) : 544862)\}. \end{aligned}$$

The goal is, by the given set *S* of labeled queries, learning a regression model *M* in such a way that for any conjunctive range query *q* on *T, M* produces the estimated selectivity value *est(q)* close to the actual selectivity value *act(q)*. As a matter of fact, no assumption about the source or the distribution of queries in the set S is being made, however the learned model *M* is expected to produce accurate estimates for queries which are represented well in the given labeled queries set *S*, that is considered as the training set. The focus of the study on the selectivity estimation is for conjunction of two or more predicates.

B. Selectivity Estimation as a Regression Problem

The usage of regression models in the selectivity estimation process is known for the potential to capture complex relationships between query features and selectivity estimates, leading to more accurate estimated values compared to more straightforward techniques. The model can adapt to changing workloads and data distributions by continuously learning from query data. The main goal for our regression models is to learn a function that maps query features (i.e., predicates) to selectivity estimation values. The input features we used for our regression model *M* are constructed hereby: With the labeled queries set S, for each query q_j in *S*, we create a tuple of 2 x *d* values such that both lower bound and upper bound of each attribute is included and is in consecutive form $\langle lb_1, ub_1, lb_2, ub_2, ..., lb_{\mathrm{d}}, ub_{\mathrm{d}} \rangle$. In this paper, we refer to these input features as range features. For each query correspondingly, the actual selectivity value - *act(q_j)* is considered as the source of the regression label. For example, the input features for queries in $\mathrm{S_{example}}$ are $\langle 0, 10, 0, 100 \rangle$, $\langle 15, 100, 2, 17 \rangle$, with regression labels 10000, 300, respectively.

C. Utilized Machine Learning Models

For regression model *M* construction, multiple types of regression techniques are used as the data can have linear or non-linear, complex distributions, and to be exact in our findings we chose to take into account all cases in general. We consider three types of regression methods: linear, neural networks and tree-based ensembles. Since neural networks and tree-based ensembles both are trained on a large dataset of queries we expect them to learn complex relationships between query features and selectivity. This flexibility enables them to handle complex data distributions more effectively than histograms [1].

- *Linear and Multidimensional Linear Regression*: As a foundation for understanding and potentially using as a baseline approach, linear regression techniques can play a valuable role in selectivity estimation. For some database systems, the relationship between the query features (i.e. attribute domain ranges) and selectivity values might showcase certain degree of linearity. For example, simple queries used to filter data based on single attribute within a given domain range. As this range gets wider, the proportion of data being returned might decrease linearly. Multidimensional linear regression can be considered when dealing with queries that involve many attributes and when the connection between the features and selectivity are primarily linear. The "dependent variable" would be the selectivity, and attribute ranges specified by the query would be the "independent variables". However, it is worth noting that real-world data often exhibit non-linearity because of the complex interactions between attributes and data distributions in the query. In cases like this, the linear regression models would lead to significant estimation errors.
- *Neural Network (NN)*: Our study employs a neural network architecture with ReLU (Rectified Linear Unit) activation functions in the hidden layers. These activation functions create a "piecewise linear" effect, efficiently dividing the query space (comprising all possible queries) into local regions [22], [23]. Within each zone, the network approximates the selectivity function as a simple linear model. This approach offers flexibility: by increasing the number of hidden layers and neurons we can create more complex divisions of the query space, potentially leading to more accurate selectivity estimates [22]. The basic NN model consists of an *input layer* made up of nodes to feed the vector of feature values as input and a single neuron *output layer* that generates the prediction value. The network can include additional neurons in the form of *l hidden layers* in, addition to the neuron in the output layer.
  - When $l = 0$, neural network is equivalent to a linear regression model.
  - When $l > 1$, we refer to the network as a deep neural network.

  For this project we utilize a neural network with fully linked layers, meaning all neurons in one layer are connected to all neurons in the subsequent layer.
- *Tree-Based Ensembles*: Techniques such as gradient boosting and random forests fall under the tree-based ensembles category. These algorithms construct multiple decision trees. As a classifier, each tree consecutively divides the query space into rectangular regions according to particular features and their values. Each leaf node or terminal point of a tree represents a unique area of the query space defined by conjunction of the ranges on input features, i.e. $(c_1 \leq lb_1 < c_2) \wedge (c_3 < ub_1 \leq c_4) \wedge (c_5 \leq ub_2 \leq c_6)$. When a new query is being inserted, it is navigated through each tree based on its features until it reaches a leaf node. The final selectivity prediction results from the aggregation of predictions of all trees (i.e. average for random forest, weighted sum for gradient boosting). By increasing the number of trees and leaves in the ensemble, more precise partitions of the query space may be made, which potentially leads to more accurate selectivity estimates.

These methods learn partitioning the query space and estimating selectivity efficiently within each region, unlike histograms that store exact values of selectivity for a predefined queries set [23]. This translates to a smaller memory footprint, making them more scalable for real-world database systems that may have large query spaces. So, when estimating selectivity with tree-based ensembles, a small number of binary search trees are being traversed; this process is typically faster than the intricate calculations that neural networks may need to perform. Nevertheless, the precise computational efficiency might be impacted by the choice of network architecture and hardware optimization.

III. Methodology

The experimental setup for method construction is crucial for building a solid base for our study. Our methodology adopts a comprehensive approach to refine selectivity estimation methods and optimize database query performance. The critical components for our approach include Python and PostgreSQL environments configuration, adherence to coding practices, selectivity estimation techniques utility, appropriate selection of regression techniques, and employed evaluation criteria.

A. Data Collection and Labels Transformation

We used three real-world datasets from different domains to evaluate the estimation quality of various realistic data distributions.

1) **Power** [24] : This dataset consists of measurements of electric power consumption in a household, sampled at a one-minute rate over almost four years. It includes various electrical quantities and sub-metering values. The dataset contains 2,075,259 measurements. For our experiment, we used 7 numeric attributes, only excluding the first two columns containing date and time values.

2) **Forest** [25] : This dataset is also known as "covertype"; it involves classifying pixels into seven forest cover types using attributes like elevation, aspect, slope, hillshade, soil type, and more. The dataset contains raw data, including binary (0 or 1) columns for qualitative independent variables like wilderness areas and soil types. The dataset has 581012 rows and 54 attributes. We filtered it, taking only the first ten numeric attributes as in [10], [12], since the other attributes in the data are binary.
3) **Higgs** [26] : This dataset addresses a classification problem aimed at distinguishing between a signal process that produces Higgs bosons and a background process that does not. The data was generated using Monte Carlo simulations. The first 21 features (columns 2-22) represent kinematic properties measured by particle detectors in the accelerator. The last seven features are functions of the first 21 features and are high-level features developed by physicists to aid in discriminating between the two classes. This scientific dataset has 11 million rows. We use only the last 7 high-level features, which are derived by physicists aiming to classify the particles correctly.

In our study, we treated each dataset as a table with valuable insights. In our study's scope, we substantiated that the reproducibility and consistency across different computing environments are ensured, streamlining the setup process. Thus, the critical first step was the configurations of Python and PostgreSQL environments adhering to established coding practices, including the use of singleton connectivity for database connections. This, in its turn, promotes optimal resource usage and code maintainability.

The primary interchange with our database is the query generation with range predicates on two or more attributes and their execution [1]. The query is considered a d-dimensional if non-trivial range bounds are present for *d* attributes. The way our query workload was constructed is the following: having a *d* dimensional dataset, we generate queries with dimensions varying from *2* to *d*, excluding one-dimensional case. With the *d* number of attributes in the particular dataset, we constructed a logic of generating queries with all possible combinations of attributes. To keep control over the high number of generating queries, we took the subsets of attributes, starting from attribute pairs to the culmination of including all attributes in a single query. Each generated query structure was run for *x* number of times, depending on the dataset size and the time taken for the generation.

For a chosen set of d attributes (A_{k_1}, A_{k_2}, ..., A_{k_d}) of a dataset, the creation of the queries was done in the following manner. For each attribute, the domain space is established and kept as a source for further steps of referral. Later, the generation of uniform random values for every attribute over the domain space has been conducted, and combined the resulting values with randomly generated signs (" $>$ "," $<$ "," $=$ "," $\geq$ "," $\leq$ "," $\neq$ "), saving them as *predicates*. As the next step, we established the integration of the predicates combination into our SELECT statement with the WHERE clause, connecting them with AND based on the defined number of dimensions in the given query.

One of the critical filtering techniques utilized was keeping only the queries that were at least returning 1 row. This helped to focus the study on realistic queries without considering all the generated queries that, in some scenarios, could potentially lead to the issue of having a majority of non-result-returning queries. Thus, before going through the execution process, we preformed the execution of the queries internally in the database, making sure to exclude the ones that do not return any rows.

The execution of each query involves an EXPLAIN ANALYZE, which displays the execution plan of estimated statement execution cost, which represents the planner's guess at how long it will take to run the statement (planning time in milliseconds, execution time in milliseconds) [27]. The execution process includes calculating selectivity estimation on each query using existing techniques such as AVI and STHoles. The resulting values later served as a comparison base for our model outcomes.

We applied the log-transformation for training labels generation, using base 2, to the selectivity value act(q_i). The final estimation, est(q_i) is obtained at estimation time by applying an inverse transformation to the model prediction. Despite the fact of transformation being simple, it makes the use of generic regression techniques possible for problem domains such as selectivity estimation without altering the technique's implementation details — and this is due to the possibility of selectivity variation across queries being enormous and relative metric is more relevant and the focus on relative error is increased. When the regression models perform well, we usually anticipate them to produce low mean-square error (MSE) across all queries. As an example, for two queries q_1 and q_2 with actual selectivity values $act(q_1)$ = 46882 and $act(q_2)$ = 552096, using these selectivity values $act(q_i)$ directly as training labels, the model could predict $est(q_1)$ = 46000 and $est(q_2)$ = 500000 rather than $est(q_1)$ = 45500 and $est(q_2)$ = 550000. It implies, this often yields high relative error for q_2 and low relative error for q_1, since the method is forced to minimize the difference between log $act(q_i)$ and log $est(q_i)$ by using log transformed labels, i.e.,

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

which should lower q-error (generally speaking, relative error).

The equation below represents the Mean Squared Error (MSE) between the actual selectivity and the estimated selectivity for each query q_i in the test set S_{test} in the log-transformed space.

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

A low value of the expression indicates that the logarithm of the estimated selectivity log est(q_i) closely matches the

logarithm of the actual selectivity log act(q_i) for most queries in the test set, which in turn means that the estimation process is accurate.

Noting that minimizing max(e_i) is equivalent to minimizing the worst-case q-error, and minimizing the mean of log(e_i) is comparable to minimizing the geometric mean of q-error [1]. So the optimization for just one of these could lead to undesirable results for another. Both objectives can be met by minimizing the mean squared log(e_i), as during the average calculation larger q-errors get more weight.

B. Comparative Techniques for Selectivity Estimation Problem

For comparison we used the techniques listed below.

1) **AVI**, is the most common and widely used technique for most database systems. As mentioned earlier in the article, it is based on the assumption of the distributions of individual attributes A_i are independent of each other regardless of the actual data dependencies [28]. The data distribution of each attribute is approximated separately using any of the one-dimensional histograms or other techniques. One of the advantages of this approach is that it is possible to use for good-quality one-dimensional histograms, which are not expensive to compute, store, and maintain. Although this is the traditional technique for selectivity estimation, it is also known that the assumption almost always provides an underestimate for correlated attributes resulting in approximate joint data distribution [29]. When the tuples of a relation of attributes are organized into groups according to their values in one of the attribute, within each group, the distribution of values in the other attribute is identical up to a constant factor [29]. This in its turn results in miscalculation and underestimation.
2) **STHoles** [9], a "workload-aware" state-of-the-art multidimensional histograms technique that is known for its effectiveness in accurate multi attribute query selectivity estimation. The histograms are constructed according to the analysis of the query results. This technique uses bucket nesting to capture regions of the data domain with close-to-uniform. Buckets are allocated where it is most needed based on the workload, which in its turn lead to accurate query selectivity estimations [9]. In general, this approach is considered to be better than heuristic assumptions and lots of multidimensional histogram techniques including AVI. Nevertheless, when having large dimensions of STHoles results in large errors, even when using huge number of buckets (i.e. 3000) and larger estimation time.
3) **KDE** [12], Kernel Density Estimator which uses feedback queries for improving the quality of estimation for any given data. The state-of-the-art for selectivity problems are multidimensional histograms, which offer good quality estimates, they have shortcomings of often being complex to construct and sometimes impossible to maintain. KDE is treated as alternative to those techniques, and it uses a random sample to calculate the selectivity values by averaging local probability distributions (also known as kernels) that are centered on the sampled items. KDE model is simple and has been proposed by several writers [10], [30] as range selectivities' estimation method as it also has statistical advantages [31]. However, currently available KDE-based selectivity estimators still need to be developed and are can hardly compete with the most advanced multidimensional histogram techniques in use. In our study we used the KDE with SquaredQ loss function and Batch variant for bandwidth optimization by the motivation of the reference article.

C. Metrics Selection

To assess the accuracy of our model M, we use a test set of queries S_{test} and *q-error* [7]. The choice of *q-error* is made based on its relativity and symmetricity, which makes it a perfect suit for our purposes [6], [13]. For each query q in the test set S_{test}, we calculate the *q-error* using the formula:

$$e_i = \max\left(\frac{\text{est}(q_i)}{\text{act}(q_i)}, \frac{\text{act}(q_i)}{\text{est}(q_i)}\right)$$

Here, $act(q_i)$ represents the actual selectivity of the query, and $est(q_i)$ represents the selectivity estimated by our model. We assume that $act(q_i) \geq 1$ and $est(q_i) \geq 1$. To evaluate the overall accuracy of our model$M$ across all the test queries S_{test}, we calculate the geometric mean of the q-error values, since it is better at handling outlier errors compare to the arithmetic mean.

D. Models Training Setup

The appropriate training setup employed to optimize the performance of the explored models (neural networks, tree ensembles, or potentially linear regression models) is the crucial step in achieving accurate selectivity estimation results. The main steps performed for the training preparation for our study has the following construction:

- Data Preparation: We loaded the dataset containing query features and corresponding labels. The feature matrix consists of lower and upper bounds of queries, and the label vector contains selectivity values.
- Regression Model Training: Several regression models were trained using scikit-learn, such as Linear Regression, Elastic Net.
- Neural Network Training: Leveraging PyTorch, we employed mini-batch gradient descent to train a neural network. The architecture included input, hidden, and output layers. We utilized the Rectified Linear Unit (ReLU) activation function for the hidden layer.
  - We compared the selectivity estimations resulting with the following models: Linear Regression, Elastic Net, Fully Connected Neural Network (FCNN), Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, Support Vector Regressor, K-Neighbors Regressor

- Model Evaluation and Prediction: We evaluated the trained models using the metrics: MSE, R^2 Score and Geometric Mean Q-Error.

To train a regression model, a collection of queries labeled with actual selectivities and optionally the Cost Estimation (CE) feature values are being used. Past query execution results serve as valuable source for collecting training data without the addition of extra expense. Regression models are similar to query-driven methods proposed previously. In order to bootstrap the method before getting external queries, we can take into account instances of generating training examples. However, collection of actual selectivities necessitates the execution of large set of queries over the data, which is resource-consuming and may take up to (#rows x #queries) operations since queries arbitrarily overlap [1]. Nevertheless, several strategies can be employed to reduce latency.

IV. Results and Evaluation

In the study, the evaluation was done on the performance of various regression models for selectivity estimation on the datasets. It was based on Mean Squared Error (MSE), R-squared (R2) score, and Geometric Mean Q-Error metrics. Based on the evaluation, we observed the following:

- MSE and R2 Score: Among the models evaluated, the Decision Tree Regressor performed the best, with the lowest MSE and highest R2 score.
- Geometric Mean Q-Error: The Linear Regression models exhibited the lowest Geometric Mean Q-Error, indicating better overall performance in terms of selectivity estimation.

Considering the overall performance metrics, the Decision Tree Regressor model fits the best for selectivity estimation problem due to its lower R and MSE scores, indicating more accurate selectivity estimation. The second best is the Random Forest Regressor with almost similar features.

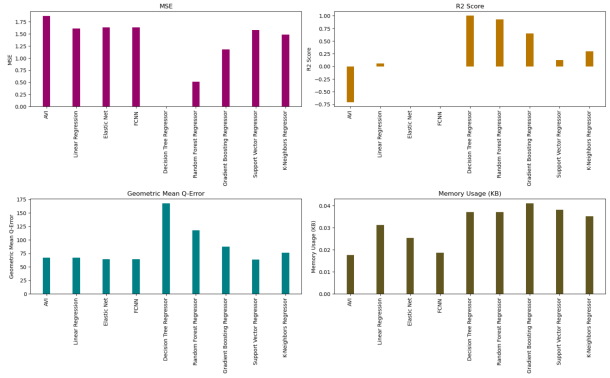


Fig. 3: Model Performance Evaluations on a dataset

V. Conclusion and Future Work

This capstone thesis explored the application of standard regression techniques to the problem of selectivity estimation of range predicates. We discovered diverse techniques available for estimation optimization and experimentally compared those techniques with regressions. We carried out in-depth analyses of multiple datasets with an emphasis on database system performance. With extensive empirical evaluation over those datasets, we showed that the accuracy of regression models is significantly better than other existing methods. We concluded that the learning based approach plays significant role for the use of database systems due to its properties such as accuracy and small memory footprint. Future work could potentially go in a number of interesting directions, in our opinion. These include the creation of combined executions of learned models and existing techniques, which could result in increased development and precision; integration and model construction on various database systems; automated handling of attribute subsets and join query handling, etc.

References

[1] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri, "Selectivity estimation for range predicates using lightweight models," *Proceedings of the VLDB Endowment*, vol. 12, no. 9, pp. 1044–1057, 2019.

[2] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine, *et al.*, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends® in Databases*, vol. 4, no. 1–3, pp. 1–294, 2011.

[3] S. Chaudhuri, "Query optimizers: time to rethink the contract?," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 961–968, 2009.

[4] "Understanding optimizer statistics with oracle database 18c." https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-stats-concepts-0218-4403739.pdf, Feb 2018.

[5] "Postgresql documentation." https://www.postgresql.org/docs/current/row-estimation-examples.html, Feb 2024.

[6] 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*, vol. 27, pp. 643–668, 2018.

[7] G. Moerkotte, T. Neumann, and G. Steidl, "Preventing bad plans by bounding the impact of cardinality estimation errors," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 982–993, 2009.

[8] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?," *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 204–215, 2015.

[9] N. Bruno, S. Chaudhuri, and L. Gravano, "Stholes: A multidimensional workload-aware histogram," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pp. 211–222, 2001.

[10] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi, "Selectivity estimators for multidimensional range queries over real attributes," *the VLDB Journal*, vol. 14, pp. 137–154, 2005.

[11] M. Shekelyan, A. Dignös, and J. Gamper, "Digithist: a histogram-based data summary with tight error bounds," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1514–1525, 2017.

[12] M. Heimel, M. Kiefer, and V. Markl, "Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1477–1492, 2015.

[13] M. Müller, G. Moerkotte, and O. Kolb, "Improved selectivity estimation by combining knowledge from sampling and synopses," *Proceedings of the VLDB Endowment*, vol. 11, no. 9, pp. 1016–1028, 2018.

[14] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil, "Leo-db2's learning optimizer," in *VLDB*, vol. 1, pp. 19–28, 2001.

[15] A. Aboulnaga and S. Chaudhuri, "Self-tuning histograms: Building histograms without looking at data," *ACM SIGMOD Record*, vol. 28, no. 2, pp. 181–192, 1999.

[16] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, "A pay-as-you-go framework for query execution feedback," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1141–1152, 2008.

[17] K. Guidolin, *Developing the Porphysome Nanoparticle for Photodynamic Therapy of Colorectal Cancer*. PhD thesis, University of Toronto (Canada), 2022.

[18] S. Lakshmi and S. Zhou, "Selectivity estimation in extensible databases-a neural network approach," in *VLDB*, vol. 98, pp. 24–27, 1998.
[19] H. Lu and R. Setiono, "Effective query size estimation using neural networks," *Applied Intelligence*, vol. 16, pp. 173–183, 2002.
[20] H. Liu, M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte, "Cardinality estimation using neural networks," in *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, pp. 53–59, 2015.
[21] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," *arXiv preprint arXiv:1809.00677*, 2018.
[22] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *Advances in neural information processing systems*, vol. 27, 2014.
[23] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1244–1253, 2018.
[24] G. Hebrail and A. Berard, "Individual Household Electric Power Consumption." UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C58K54.
[25] J. Blackard, "Covertype." UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50K5N.
[26] D. Whiteson, "HIGGS." UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5V312.
[27] "Postgresql documentation_2024a." https://www.postgresql.org/docs/current/sql-explain.html, Feb 2024.
[28] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das, "Multi-attribute selectivity estimation using deep learning," *arXiv preprint arXiv:1903.09999*, 2019.
[29] V. Poosala and Y. E. Ioannidis, "Selectivity estimation without the attribute value independence assumption," in *VLDB*, vol. 97, pp. 486–495, 1997.
[30] B. Blohsfeld, D. Korus, and B. Seeger, "A comparison of selectivity estimators for range queries on metric attributes," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pp. 239–250, 1999.
[31] D. W. Scott, *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.