Query-Driven Learning for Automating Exploratory Analytics in Large-scale Data Management Systems

Supervisors:
Dr. Christos Anagnostopoulos
Prof. Peter Triantafillou

Fotis Savva

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computing Science

College of Science and Engineering
University of Glasgow

December 13, 2020

Abstract

As organizations collect petabytes of data, analysts spend most of their time trying to extract insights. Although data analytic systems have become extremely efficient and sophisticated, the data exploration phase is still a laborious task with high productivity, monetary and mental costs. This dissertation presents the Query-Driven learning methodology in which multiple systems/frameworks are introduced to address the need of more efficient methods to analyze large data sets. Countless queries are executed daily, in large deployments, and are often left unexploited but we believe they are of immense value. This work describes how Machine Learning can be used to expedite the data exploration process by (a) estimating the results of aggregate queries (b) explaining data spaces through interpretable Machine Learning models (c) identifying data space regions that could be of interest to the data analyst. Compared to related work in all the associated domains, the proposed solutions do not utilize any of the underlying data. Because of that, they are extremely efficient, decoupled from underlying infrastructure and can easily be adapted. This dissertation is a first account of how the Query-Driven methodology can be effectively used to expedite the data exploration process focusing solely on extracting knowledge from queries and not from data.

Acknowledgements

Doing a PhD has been a long and fun journey, with moments of despair and moments of joy. It would not have been possible to complete this journey without my supervisors Dr. Christos Anagnostopoulos and Prof. Peter Triantafillou. I would like to express my gratitude towards them as they helped me pursue what I found interesting and did not impose any restrictions on me. Special mention to Prof. Dimitrios Pezaros who gave me the opportunity to work in his lab for the past months and helped me expand my knowledge in the domain of networked systems.

I would also like to thank all the people in my lab Drs. Jing Wang, Foteini Katsarou, Nikos Ntarmos for all the interesting conversations and support throughout my first years. Special thanks goes out to Wei Ma and Dr. Atoshum Samuel Cahsai, for their support, friendship and feedback. To all of the people in the Department of Computer Science that we crossed paths with, I would like to thank you for all the advice and help whenever I needed it.

Finally, I would like to thank Anastasia for her kindness and making my life interesting for the past few years. It would all have been impossible without the support of my family, Alexis and Demetra my father and mother who helped shape me and my younger brother and sister, Kostas and Georgia for all the incredible moments we spent together growing up. They all helped me become who I am today, I owe it all to them.

Table of Contents

Chapter 1

Introduction

1.1 Overview & Contributions

1.1.1 Motivation

As data storage capacity has become cheaper and larger, organizations have switched to data-driven decision making. By having a much larger capacity to store data, such as financial transactions, machine logs and unstructured data like images/audio, decisions taken by large organizations have become more informed. However, a number of challenges have to be addressed before being able to reach decisions based on vast amounts of data. First, data have to be ingested through a collection process and initially stored in a (possibly distributed) filesystem. Then, through a process that is commonly abbreviated as ETL (Extract-Transform-Load), the data are pre-processed and loaded into a Data Warehouse. A Data Warehouse can be any kind of database system that executes queries issued by data analysts and perform data analysis. The complete process is shown in Figure 1.1.

Each step of the process is time-consuming, but the most time-consuming part is generating insights and performing data analysis. This process is indicated by the red arrow in Figure 1.1. The processes that take part, before data analysis can even begin (like data collection an ingestion), are tasks that have to be performed a single time. On the other hand, data analysis is a long iterative process. It requires the presence of a data analyst sitting on the other end, issuing queries to the Data Warehouse, waiting for responses and then based on those responses, moving on to generate business decisions.

Hence, developing methods that would help expedite this last part of the process could be of tremendous benefit. The crux of our methods that will be described later lie in the use of Machine Learning (ML) to automate a number of processes. This is inline with recent developments in self-tuning databases [133]. In this thesis we similarly exploit recent advances

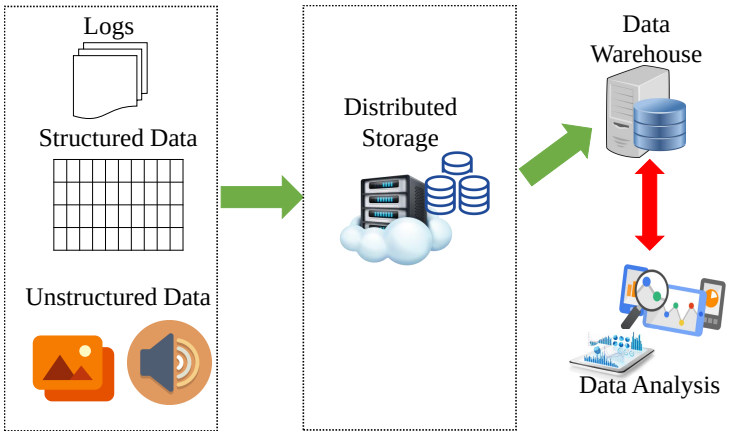

Figure 1.1: The process of collecting heteregeneous data and loading them into a Data Warehouse. Data Analysis is conducted by repetitively querying the Data Warehouse.

in ML and show how traditional relational database systems can be augmented and provide advanced features. The approaches described in this thesis are not focused on auto-tuning database systems with ML, like in [133]. Instead, we show how complex ML models can be deployed to help any kind of data analytic system during the data exploration phase with a focus on data analysts. This thesis is one of the first of its kind in describing such techniques which seem to be the future as we are seeing more and more works fusing together ML and DBMSs [14, 15]. Over the next few sections and chapters we discuss how we were able to achieve this, the numerous challenges that we had to address along the way and the lessons that we have learned.

1.1.2 Solution Overview

We focus on developing methods/systems that can expedite the data analysis process. The solutions proposed in this thesis are focused on three pilars, which we abbreviate as the three '**E**'s and are divided into three core chapters. The three pillars are **E**stimation, **E**xploration and **E**xploitation. An overview of all the solutions is shown in Figure 1.2.

The first part of the proposed solution is shown in the left-most part of Figure 1.2. In this case, an analyst issues a query to a Data Warehouse and receives the response 140. A common example of such a query could be *"How many restaurants are within a given radius ?"*[1]. The response, computed by the Data Warehouse, could be returned to the user in seconds or minutes, depending on the current load that the system is experiencing and the amount of data. What we propose is an alternative mechanism that can provide an estimate of the response at a much faster rate. Our mechanism would have nothing to do with the Data Warehouse or any data stored in it. So, it will be able to compute an approximate answer,

[1]Typically data analysts will issue more complex queries, but we provide a simple example for clarity.

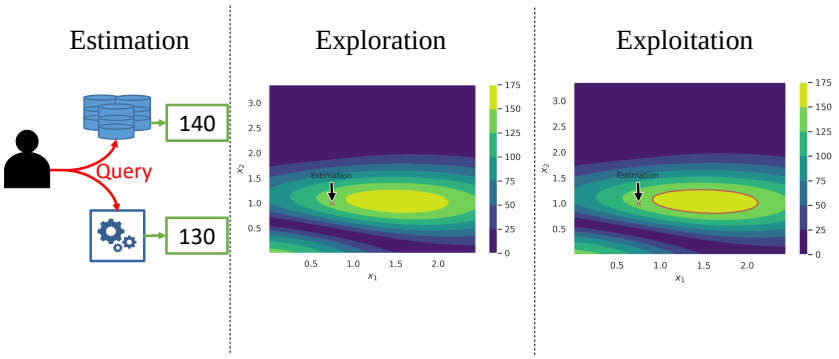

Figure 1.2: The three 'E's : (Left) Estimation: An alternative mechanism that can estimate the answer of a query much faster. (Middle) Exploration: An estimate, is depicted as a point in a much larger landscape, which can be mapped and used for exploration (Right) Exploitation: The task is to pinpoint regions in the mapped landscape.

130, that is close to the true answer, 140, at a fraction of the time required by the actual system. In an iterative process, such as data analysis, receiving approximate answers at a much faster rate can help analysts explore data more efficiently.

The second part of our proposal, shown in the middle of Figure 1.2, has to do with **E**xploration. Imagine that Figure 1.2(middle) shows the landscape of all possible responses, that the query *"How many restaurants are within a given radius ?"* could yield, given that we change the query parameters ie *location* and *radius* which are the two dimensions shown in the figure. The previous response, 140, is shown as a single point in this figure. Now, given a method, that can inform, how these responses shift, depending on the variation in the query parameters, data analysts can choose better queries to issue. To put it simply, the **E**xploration pillar, is about providing a *map* that shows the query parameters which yield a higher number of restaurants. This way, the analyst can more easily navigate towards this location.

The third and last part of the proposed solution is depicted as the right-most part of Figure 1.2. It is associated with the **E**xploitation pillar. Given the first two pillars, **E**stimation and **E**xploration, and the example we have used so far, it is natural to ask *"Show me the regions with the most restaurants."*. In Figure 1.2(right), the result of the first pillar, is shown as a single estimated response to the question *"How many restaurants are within a given radius ?"*. This is visualised within the landscape of all possible values that we were able to map by addressing **E**xploration. For **E**xploitation we have developed an algorithm that is able to pinpoint regions within this landscape that satisfy a user constraint. The region circled in red, shown in Figure 1.2(right), informs the analyst which query parameters would generate an interesting result. By using this method, the analyst can concentrate their focus on smaller

regions (focusing on less tuples in the database), without even issuing a query. This can potentially reduce the time spent in the iterative process of data analysis, as it might reduce the number of iterations needed, to reach a possible outcome or insight.

1.1.3 Academic Contributions to the Research Community

The work presented in this thesis is largely based on published work by the author and other collaborators. The chapters and the associated papers to which the chapters are based on are listed below.

1. Chapter 2: Query-Driven Learning for Estimation of Aggregate Answers
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"ML-AQP: Query-Driven Approximate Query Processing based on Machine Learning."* arXiv preprint arXiv:2003.06613 (2020). [116]
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"Adaptive learning of aggregate analytics under dynamic workloads."* Future Generation Computer Systems (2020). [115]
    - **Savva, Fotis**. *"Query-Driven Learning for Next Generation Predictive Modeling & Analytics."* Proceedings of the 2019 International Conference on Management of Data. (SIGMOD SRC) 2019. *Recipient of* 2^{nd} *place.* [112]
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"Aggregate query prediction under dynamic workloads."* 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019. [114]
2. Chapter 3: Query-Driven Explanations for Exploratory Analytics
    - **Savva, Fotis**, Christos Anagnostopoulos, Peter Triantafillou, and Kostas Kolomvatsos. *"Large-scale data exploration using explanatory regression functions."* ACM Transactions on Knowledge Discovery from Data (2020). [118]
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"Explaining aggregates for exploratory analytics."* 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018. [113]
3. Chapter 4: Identifying interesting subspaces with Query-Driven Surrogate Models
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"SuRF: identification of interesting data regions with surrogate models."* 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020. [117]

4. Chapter 5: Dynamic Data & Query Workloads Adaptation

    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"ML-AQP: Query-Driven Approximate Query Processing based on Machine Learning."* arXiv preprint arXiv:2003.06613 (2020). [116]
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"Adaptive learning of aggregate analytics under dynamic workloads."* Future Generation Computer Systems (2020). [115]
    - **Savva, Fotis**, Christos Anagnostopoulos, and Peter Triantafillou. *"Aggregate query prediction under dynamic workloads."* 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019. [114]

The author of this thesis was extensively involved in the publication of all of the aforementioned works which included tasks such as : idea conceptualization, preliminary evaluations, code development, paper writing, literature review, design/development of proposed algorithms and models, review rebuttals and addressing of reviewer comments, data analysis etc.

1.1.4 Thesis Structure

The thesis is divided into five large parts. The *Introduction*, provides an overview of the topic discussed in the rest of the chapters and also summarises some of the key contributions made to the research community. The second part of the *Introduction*, introduces some of the concepts that are used in later chapters. Chapters 2,3,4 are the core chapters of this thesis. Specifically, Chapter 2, is devoted to the first 'E' which is *Estimation*, Chapter 3 to the second 'E', *Exploration* and Chapter 4 to *Exploitation*. In Chapter 5, we discuss approaches that could be used in cases where the models, described in the core chapters, need to be adapted in cases where data and workloads change. Chapter 6, provides some concluding remarks and limitations, which also serve as ground for future work.

1.2 Background

This section is dedicated to introducing some of the concepts that might be unfamiliar to the reader. The aim is to provide a brief overview of techniques used and referred to throughout this thesis. Consequently, it will be much easier to understand the contexts to which they are applied to. Firstly, a quick overview of ML is given, and then we contrast this with the methodology of Query-Driven Learning (QDL). Finally, we provide an overview of the *Analytics* landscape encompassing some of the proposed systems to be used as well as common exploration techniques.

1.2.1 Machine Learning

ML is a generic term, that includes a plethora of methods/algorithms that teach machines how to learn from data. These methods are grouped into different categories such as *Supervised learning* or *Unsupervised learning* which are generally distinguished by the associated learning task. In this thesis, we make use of both *Supervised* and *Unsupervised* learning. Specifically, we use *Supervised Regression* algorithms and *Unsupervised Clustering* algorithms. *Clustering* and *Regression* refer to specific tasks that are accomplished by *Unsupervised* and *Supervised* learning respectively.

Supervised Regression

In a *Supervised Regression* task, the objective is to predict a real value $y \in \mathbb{R}$ given a vector of inputs $\mathbf{x} \in \mathbb{R}^d$. Normally, we have input vector $\mathbf{x} = (x_1, \ldots, x_d)^\top$, where the different dimensions of vector $\mathbf{x}$ denote different *features*. Given a collection of input vectors $\mathbf{x}$ and response variables y, $\mathbf{X} \in \mathbb{R}^{N \times d}$, $\mathbf{y} \in \mathbb{R}^N$, *Supervised Regression* algorithms learn a function $f : \mathbb{R}^d \to \mathbb{R}$, using a *training procedure*. The training procedure usually varies by the type of algorithm but generally the aim is to minimize a variant of the Expected Prediction Error (EPE). So the objective is to derive a model that sufficiently minimizes EPE which is defined in (1.1).

$$\mathcal{M}* = \arg\min_{\mathcal{M}*} \sum_{i}^{N} (y_i - y_{i,\mathcal{M}*})^2 \tag{1.1}$$

Where $y_{i,\mathcal{M}*}$, is a prediction given by the model. A typical model, to make things more intuitive, is a simple *Linear Regression* model. The structural form of *Linear Regression* is given in (1.2):

$$y = \mathbf{w}^\top \mathbf{x} + \epsilon \tag{1.2}$$

Where, $\epsilon \in \mathcal{N}(0, 1)$ is an irreducible error and $\mathbf{w} \in \mathbb{R}^d$ is a vector of coefficients that are estimated using a training procedure and EPE. Multiple training procedures exist that also vary by the variant of EPE that is being minimized and on whether there exists a closed-form solution with respect to the model. In the end of the training procedure vector $\mathbf{w}$ holds the coefficients that minimize EPE. So redefining (1.1), we obtain $\mathbf{w}$ by (1.3)

$$\mathbf{w}* = \arg\min_{\mathbf{w}*} \sum_{i}^{N} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \tag{1.3}$$

The coefficients, $\mathbf{w}$, or any other parameter that a model might have are estimated using a subset of the dataset $(\mathbf{X}, \mathbf{y})$. The dataset is separated into two subsets, where $(\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}})$ is the *training* dataset and $(\mathcal{X}_{\text{test}}, \mathcal{Y}_{\text{test}})$ is the *testing* dataset. With $|\mathcal{X}_{\text{train}}| > |\mathcal{X}_{\text{test}}|$ and sub-

sequently $|\mathcal{Y}_{\text{train}}| > |\mathcal{Y}_{\text{test}}|$ and where $|\cdot|$ indicates the size of a set. Any ML algorithm, goes through a training procedure in which the *training* subset is used to estimate the values associated with all parameters of the model.

Once, the training procedure is over, the model's performance is measured against the *testing* dataset. Any subsequent prediction, that we wish to perform to infer an unknown label y, is obtained using the model that was estimated by the training procedure. Hence, in the case where we can obtain a vector of features $\mathbf{x}$ but we do not know its associated label y we can predict it using the estimated model. This is shown in (1.4), where the estimated coefficients, $\mathbf{w}_*$, for *Linear Regression*, are used to predict the output label y_{new} of a *new* vector of inputs $\mathbf{x}_{\text{new}}$.

$$y_{\text{new}} = \mathbf{w}_*^\top \mathbf{x}_{\text{new}} \tag{1.4}$$

Unsupervised Clustering

An *Unsupervised* algorithm differs from a *Supervised*, in that no labels y exist for the given dataset. Thus, *Unsupervised* algorithms extract patterns solely based on a dataset $\mathbf{X}$. In an *Unsupervised Clustering* task the objective is to identify clusters within the dataset. These clusters will contain vectors $\mathbf{x}$ that are more 'similar' with each other than with vectors belonging to another cluster. Formally, similarity in this setting can be defined as the squared euclidean distance $||\mathbf{x}_i - \mathbf{x}_j||_2^2$ between two vectors. Squared Euclidean distance between two vectors is obtained by (1.5).

$$||\mathbf{x}_i - \mathbf{x}_j||_2^2 = (x_{1,i} - x_{1,j})^2 + (x_{2,i} - x_{2,j})^2 \ldots + (x_{d,i} - x_{d,j})^2 \tag{1.5}$$

The identified clusters are disjoint, in that clusters do not overlap i.e two clusters cannot contain the same vectors. The clusters are identified by generally minimizing the criterion in (1.6):

$$\sum_{i}^{n} \min_{\mu_j \in C} (||\mathbf{x}_i - \mu_j||_2^2) \tag{1.6}$$

Where C contains cluster-representatives, with each μ_j being the cluster-represenative for the j^{th} cluster. In K-means, the cluster-representatives μ_j are the mean vectors computed by the vectors $\mathbf{x}$ belonging to a certain cluster.

1.2.2 Query-Driven Learning

QDL is a methodology similar to *Supervised Regression*. Using QDL, models can learn to predict the results of queries by leveraging previously executed queries. Queries, issued over

any Database Management System (DBMS) or raw data, instruct the underlying system to fetch particular data points. In the case of an Aggregate Query (AQ), the data points are manipulated to produce a response. Common AQs include operators such as `COUNT`, `SUM`, `AVG` etc. Hence, we can think of AQs as queries that use Aggregate Function (AF)s, which map data points $\mathbf{a} \in \mathbb{R}^d$ to results y. Hence, each AF, is a function $g : \mathcal{A} \rightarrow \mathbb{R}$. An example is shown in Figure 1.3, where an AQ, first defines which data points are going to be retrieved and then through an AF maps those data points to a result.

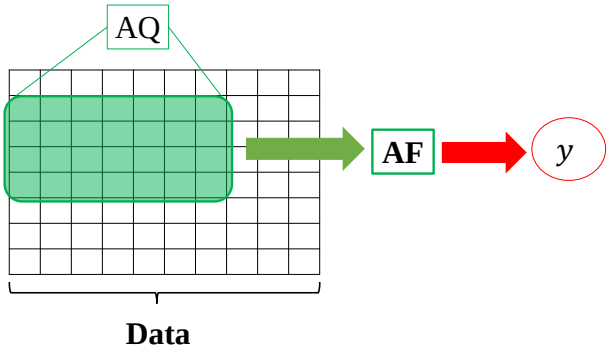

Figure 1.3: An AQ selecting a subset of the complete dataset $\mathcal{A}$ and then through an AF mapping those data points to a scalar result y.

Whenever, an AQ is executed, it can be logged and stored for monitoring purposes. Although, the logged queries cannot provide the individual data points accessed, we can observe the parameters that were used. Under a relational schema, these parameters would be : (i) predicates that were used (ii) tables that were used (iii) the AFs that were used etc. Hence, in QDL we make use of these parameters, available for many queries, to identify a function that is analogous to an AF. The only difference, is that the identified function f, maps query parameters $\mathbf{q}$ to results y,.

Having a set of queries and their responses, $\mathcal{Q} = (\mathbf{q}, y)$, QDL uses ML to identify models that can sufficiently reproduce the responses that would have been produced, given that the query was executed. By using QDL, we hope to alleviate some of the inherent computational complexity in computing the responses of AQs. This fact, introduces many opportunities, to expedite several data exploration tasks. Over the next few chapters we show how QDL can be used to :

1. Provide an inexpensive backend analytics systems that can compute approximate answers to any AQ
2. Explain data spaces by interpretable ML models that can help navigate data analysts during data exploration

3. Identify regions of interest within data spaces in polynomial time. A task that is otherwise of exponential complexity.

Each one of the aforementioned contributions, comes with its own challenges addressed in the individual chapters.

1.2.3 Analytics & Data Exploration

The contributions made in this thesis fall under the wide umbrella of *Data Analytics & Data Exploration*. Multiple systems were invented to assist large scale data analysis [23, 38, 144, 57] as organizations have rapidly shifted into data-driven decision making. Hence, *Data Analytics* covers a wide spectrum of operations that are tightly coupled with data-driven decision making. From operations such as data cleaning, data ingesting, data storing, data manipulation to operations that facilitate the analysis of data like visualisations, predictive modelling etc.

A core part of the *Data Analytics* landscape is *Data Exploration*. It is at this part in the data analytics pipeline that analysts gain an understanding of the data. *Data Exploration* includes the repetition and refinement of queries [66] until some conlusions can be drawn about the data. During *Data Exploration*, multiple AQs could be issued which make the process long and unproductive. In the following chapters of this thesis we show how this part of the process could benefit from approximate answers [51, 99, 12, 70].

In addition, during *Data Exploration*, analysts might stumble upon results that may need further explanation. Prior work [43, 108, 91, 27, 140] showed how such explanations can be given to analysts. But generating explanations is a time-consuming process. In addition, explanations in prior work were given in a one-off basis, in that no additional benefit was clear, by an explanation that explained the output of a given query. The work described in this thesis elaborates on an approach that can efficiently provide explanations that explain AQs with the goal of assisting exploration by generating insights.

Finally, an important objective during *Data Exploration* is to rapidly identify data regions that could be interesting [50, 120, 25, 24]. This is an extremely laborious task, which often includes a tight coupling of repetitive visualisation and use of exhaustive algorithms that repeatedly issue queries at a data analytics backend system. In Chapter 4, we describe an automated mechanism, based on QDL, that expedites the process of identifying interesting regions and *Data Exploration* in general.

Chapter 2

Query-Driven Learning for Estimation of Aggregate Answers

2.1 Introduction

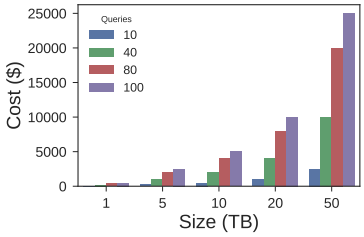

Figure 2.1: Costs associated with using cloud-managed databases (BigQuery). The x-axis is the amount of data used per query and the y-axis is the associated costs with the average number of queries daily.

Due to an increase in data volume and the adoption of data-driven decision making, organizations have been struggling to process and store data efficiently. Hence, organizations have been turning to popular Cloud providers that have created large-scale Data Warehouse solutions [38, 111, 57] able to store and process large quantities of data. However, the problem still remains, as multiple queries are issued by multiple analysts which can often overburden a cluster and carry a monetary cost. For instance, looking at Figure 2.1, we can observe the increasing costs associated with an increasing data size. The associated cost (in y-axis) is obtained after multiplying the cost of scanning certain amount of data (in x-axis) with a varying number of queries shown as colored bars [1]. This might be a prohibitive cost for organizations which are looking to accumulate petabytes of data in the near future. In addition,

[1] Data For this graph were obtained from : `https://cloud.google.com/bigquery/pricing`

data analysts want to extract information without significant delays. It has been shown that data analysts' productivity can be affected by long response times. This effect is described as the *interactivity constraint*[85] which is a limit (around 500ms) on the maximum response time that can be experienced before productivity is negatively affected. Thus, there is a dire need for systems that could mitigate the costs while offering interactive response times.

In this chapter we introduce ML-AQP, an abbreviation of Machine Learning for Approximate Query Processing, a system that leverages QDL to save computational and monetary resources whilst offering interactive response times. In short, ML-AQP uses previous queries, executed at a Data Warehouse, to train ML models that predict the responses of future queries. Hence, cost is mitigated as queries no longer have to be executed by Data Warehouses. Instead, analysts can issue queries to ML-AQP, which uses the trained models to predict their answers. This operation carries little to no cost, as the sole operation is the prediction performed by a model. The prediction is generated at a fraction of the time required by the Data Warehouse to produce an exact result, guaranteeing interactive response times.

Concretely, this chapter can be summarized into the following points:

- ML-AQP makes use of a flexible vectorized representation for (SQL) queries. This is a crucial step before training and using ML models.
- ML-AQP is a light-weight mechanism offering aggregate query result estimation with low **storage**, **computational**, **monetary** overhead.
- Query-Driven AQP engines are on average $3\times$ faster than sampling-based AQP engines, with $116\times$ less memory footprint and $100\times$ less preprocessing (model training) time.
- Probabilistic error guarantees, based on Quantile Regression, complement the approximate answers given by ML-AQP.
- All AFs (including AFs like MIN/MAX which have been difficult to estimate by traditional sampling methods) are supported by ML-AQP.
- A comprehensive performance evaluation using synthetic and several real-world datasets and workloads which substantiate performance claims are presented at the end of this chapter.

All of the aforementioned points are discussed in the following sections of this chapter.

2.2 Background & Related Work

In this section we introduce some prior work that inspired the creation of ML-AQP along with recent developments of applying ML to Data Management problems. We highlight the core differences, of existing techniques and ML-AQP and how they complement each other.

To meet the needs of interactive query processing in large analytic environments various big data engines [23, 38, 144] and columnar databases [57] have been developed. These works have given the ability to efficiently process large quantities of data over distributed environments. However, the goal of truly interactive analysis still remains elusive, as such engines still have to examine large amounts of data, often resulting in moving data over the network, spilling partial results to disk and performing complex computations.

Interactive response times are needed within the context of *exploratory analysis*. This is a task performed by data analysts [66] to better understand various data sets and includes visualising different variables in a dataset, executing aggregate queries, comparing cohorts, building predictive models and more. It is an invariable step in the process of further constructing hypotheses or training predictive and inferential models to answer business questions.

During *exploratory analysis*, interactive responses are crucial, as long waiting times might block an analyst's train of thought. An interesting observation, is that an approximate answer to exploratory queries, is often enough to the analyst. For instance, when comparing the number of records, within multiple groups, approximate results that return the same boolean value for a condition are adequate. Formally, let $y_1 \leq y_2 \leq \dots y_g$, be the ordered results of multiple groups, with each y_i signifying the number of records contained in the i^{th} group. If we can obtain approximate results $\hat{y_1} \leq \hat{y_2} \leq \dots \hat{y_g}$, adhering to the same order, then the analyst can reach to the same conclusions. An approximate answer, can be retrieved much faster than an exact answer, thus allowing the exploration process to continue without any mental blocks.

Various techniques have been developed over the years, to produce approximate answers to queries, requiring only a fraction of the time needed by actual systems. Such systems are commonly called Approximate Query Processing (AQP) systems [51, 99, 12, 70]. By trading off some of the accuracy they allow for order of magnitude speed-ups in execution.

Research in AQP has been strong the last decades [145, 104, 95, 10, 99, 98, 12, 70, 69, 63, 34, 26, 86, 51, 64] and still the list is not exhaustive. Most of the proposed solutions require large samples [12, 98, 95, 99, 70]. We can abbreviate these solutions as sampling-based AQP engines. Sampling based AQP engines create samples over some (or all) of the columns in tables and produce answers with error guarantees based on samples.

Other approaches, such as [145, 63] perform online aggregation. These systems, produce

an initial result by processing a small amount of data. The answer is then further refined as more and more data are processed, online, until the user halts query execution.

However, a main drawback of both sampling-based and online aggregation AQP engines is that they have to reside in the same Cloud system as the data warehouse which makes them costly to maintain, as every operation carries a cost. Hence, what we propose in this chapter is a *complementary system* to that of existing AQP engines, that relies only on small ML models, that can be trained using Cloud services and later deployed centrally as a service or locally at analysts' machines.

More recently, we see the application of ML models for tackling various data management problems such as : (a) selectivity estimation [40, 75, 21, 19, 96, 74, 127, 143, 138], (b) query optimization [88], in which ML is used to decide on a query plan, (c) or to create indexes [77], for expediting visualisation [136] and (d) to AQP [86, 64, 130, 81]. We believe that this approach can be fruitful, if used with care. This is why we are not aiming to replace already existing AQP engines or data analytic systems and instead provide an addition to the stack. We believe the user needs to have a choice considering the trade-offs between speed and accuracy. Therefore, our approach is similar to the recent trend of applying ML over data management problems, in that we employ ML models for AQP. Approaches such as [96, 75, 74, 127, 143, 138] make use of ML to estimate cardinalities for their use in a query optimization setting. In ML-AQP, a similar methodology to the above works is followed. Specifically, features are extracted either from data or from queries and subsequently used as training examples for ML models. This is a standardised procedure, as any relevant work that makes use of supervised ML needs to follow this process. However, all of the aforementioned works focus on cardinality estimation and not on AQP. In addition they follow different modelling/vectorization for the queries, use different ML models and none of the works explicitly address data/workload updates [2] and error guarantees.

Compared to other approaches focusing on ML for AQP such as [86, 64, 130, 81] ML-AQP neither learns from data nor uses data to construct samples or models. ML-AQP employs a novel query-driven method, based on vectorized representations of previously executed queries and their results and is oblivious to the underlying data distribution. In addition, ML-AQP's focus is not solely in COUNT/SUM/AVG as most works [64, 130, 81], but also offers support for any kind of AF through its AF agnostic methodology. These type of aggregates are commonly divided into three categories : (a) distributive: MIN, MAX, SUM, COUNT, (b) algrebraic: AVG, and (c) holistic : MEDIAN [54]. Using our methodology all three categories can be supported. Nevertheless, data-driven approaches are surely a promising avenue and we believe all of these approaches could complement each other. In cases where: (i) data sets are massive, (ii) no models or samples can be built and stored efficiently, and

[2] Chapter 5 is dedicated to data/workload updates.

(iii) there is a low cost requirement, ML-AQP is more favorable.

2.3 Preliminaries and Supported Queries

A foremost obstacle in this endeavor is a valid representation for queries such that an ML model can associate the representation with the results obtained. If we consider every Cloud analytics system (both relational and non-relational) as a black box then, essentially, queries are executed over sets of multi-dimensional points. A single row, with multiple attributes in a table can be considered as a multi-dimensional point, to which a number of operations are performed to return a result. Both *non-relational* and *relational* database systems can be considered as large collections of attributes either grouped in a collection of normalized tables or being part of a single de-normalized dataset. We can store our data in either of the two settings and the result of a query will still be the same wherever it is processed. Figure 2.2 shows an example of this, in which data are stored under different formats. If queries were to be executed against any of the shown formats, the results would not differ. Only the way of performing data manipulation and aggregation differs. In the remainder of this section, we demonstrate how common operations in a relational schema can be performed using our proposed representation. This is without loss of generality to any kind of data-storage & processing system and it is merely used as it is widely popular and should be familiar to the reader.

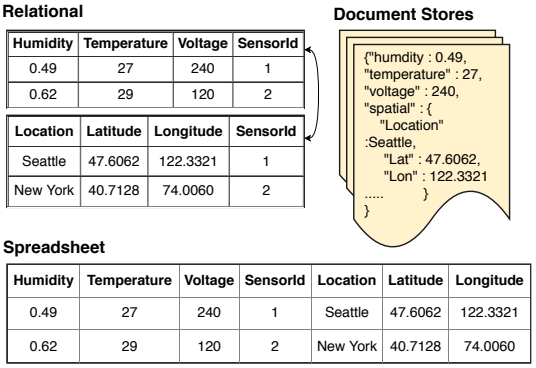

Figure 2.2: Data stored in different formats have no effect on the result returned by a query.

We first outline some definitions that will prove useful as we proceed to explain our proposed solution.

Definition 2.3.1. (Data & Attributes) As we are unaware of the underlying data storage format, we adopt a generic assumption of all data being collections of attributes, where a dataset $\mathcal{B}$ is a collection of real-valued d-dimensional vectors $\mathbf{a} = (a_1, \ldots, a_d)$, such that $\mathcal{B} = \{\mathbf{a}\}_{i=1}^{n}$. A vector $\mathbf{a}$ holds values for d attributes.

Definition 2.3.2. (Aggregate Functions) AFs are applied to the returned result-set and map a set of values to a scalar result $y \in \mathbb{R}$. An AF can be applied to a specific attribute; AFs commonly include functions such as `COUNT`, `AVG`, `SUM`. They are typically used in an SQL-style query along with various predicates and joins. Formally, an aggregate function g is a mapping from a sub-set of data set $\mathcal{B}$, $\overline{\mathcal{B}} \subseteq \mathcal{B}$, $g : \overline{\mathcal{B}} \rightarrow \mathbb{R}$ and in turn $g(\overline{\mathcal{B}}) = y$.

Definition 2.3.3. (Predicates) Predicates are used to restrict the number of rows (data vectors) returned by a query. Predicates can be considered as a sequence of negations, conjunctions, and disjunctions ($\neg, \vee, \wedge$) over attributes with equality and/or inequality constraints ($\leq, \geq, =$). A well known predicate is the *range-predicate*. A *range-predicate* effectively restricts an attribute a_i to be within a given range [lb, ub] with $a_i \geq lb \wedge a_i \leq ub$. To effectively model a sequence of predicates, we assign two *meta-attributes* for each attribute a_i and consider every predicate as a range-predicate. In the case of a range predicate the two meta-attributes are equal to the [lb, ub] of a range-predicate. For instance, without loss of generality, assume the three following predicates applied on a dataset with a single attribute a_1: (1) $a_1 \geq lb$, (2) $a_1 = c$, where c is a numerical value, and (3) $a_1 \geq lb \wedge a_1 \leq ub$. We construct two meta-attributes for each case as follows: (1) $(a_{1,lb}, -)$, where $-$ could be set to `NULL` and $a_{1,lb}$ is the supplied lb value, (2) $(a_{1,c}, a_{1,c})$, where $a_{1,c} = c$, and (3) $(a_{1,lb}, a_{1,ub})$.

Given these definitions we now explore how we can create vector representations for a variety of queries expressed in a relational setting.

2.3.1 Transforming Aggregate Queries to Vectors

SPA Queries

We first consider Selection-Projection-Aggregate (SPA) queries, in which a single aggregate is the result of a query. An SPA query operates on a single relation and might have multiple predicates. Given our definition of predicates, we obtain a meta-vector which encapsulates all the constraints $\mathbf{m} = (a_{1,lb}, a_{1,ub}, \ldots, a_{d,lb}, a_{d,ub})$ across all attributes. Hence, each SPA query can be represented by a meta-vector $\mathbf{m} \in \mathbb{R}^{2d}$ in the $2d$-dimensional real space. For all attributes that are part of the data set but not part of the query we leave the values of their associated meta-attributes as `NULL`. For instance, a simple SPA aggregate query applied over a dataset $\mathcal{B}$ with attributes $\mathbf{a} = (a_1, a_2, a_3)$, is the following:

`SELECT` AF(a_i)
`FROM` $\mathcal{B}$
`WHERE` $a_1 \geq x_1 \wedge a_2 \leq x_2$

The resulting meta-vector in the case of the query shown above would be $\mathbf{m} = (x_1, \emptyset, \emptyset, x_2, \emptyset, \emptyset)$. Where the symbol for an empty-set, $\emptyset$ could be replaced with value `NULL`.

SPJA Queries

An SPJA query is similar to an SPA query, with the sole difference being the inclusion of a `JOIN` in an SPJA query. The previous SPA query is used to present an example of an SPJA query:

`SELECT` $AF(a_i)$
`FROM` $\mathcal{B}$, $\mathcal{A}$
`WHERE` $\mathcal{B}.a_j = \mathcal{A}.a_j$ `AND` $a_1 \geq x_1 \wedge a_2 \leq x_2$

In this example, a new set $\mathcal{A}$ is introduced and joined with set $\mathcal{B}$ using a common attribute a_j. To effectively model Selection-Projection-Join-Aggregate (SPJA) queries we first redefine what it means to join two or more tables together from a query representation perspective. We assume an architecture in schema design where, if multiple tables exist, then these tables are made up of a large *fact* table along with much smaller *dimension* tables. This is widely accepted in the literature [99, 12, 59]. Specifically in a designed AQP system by Google [59], it is explicitly mentioned that all queried relations are pre-joined so that JOINs are not performed at query runtime. As a result, a data analyst simply queries the large fact table using *equi*-joins whenever they wish to project more attributes to the result set. Therefore, SPJA queries, simply increase the dimensionality of the initial row (data vector) obtained from the *fact* table. Formally, let a sub-set $\overline{\mathcal{B}}$, be the obtained sub-set of $\mathcal{B}$, after applying the predicates p included in an SPA query. If we were to re-issue an SPJA query with predicates p, joining a number of dimension tables, then the obtained sub-set $\mathcal{H}$, has the same cardinality as the sub-set of the original SPA query $|\mathcal{H}| = |\overline{\mathcal{B}}|$. This is because they would still contain the same number of vectors $\mathbf{a}$, with the dimensionality of vectors belonging to set $\mathcal{H}$ being larger. However, it is evident that the result set is still only affected by the predicates in the selection. Assuming, *equi*-joins and that the number of rows is not affected by the resulting join our initial representation is able to operate without any changes.

GROUP-BY Queries

Supporting `GROUP-BY` queries is crucial for data analytics, as analysts often issue queries to explore differences between cohorts via grouping. `GROUP-BY` queries, group vectors $\mathbf{a}$ that have the same value for a specific attribute a_i. A `GROUP-BY` query is an application of the same AF onto different sub-groups defined by that specific attribute. Hence, for an attribute a_i, we use a `DISTINCT` operator to identify the different groups (as the unique values for a_i denote different groups) and subsequently use the vectorization process of an SPA query on the individual groups. The result of the `DISTINCT` operator would be a set of group values $\mathcal{G} = (G_1, \ldots, G_k)$, which can be used with an equality predicate to construct k different queries. An example of this is shown in Figure 2.3. An SQL query with a

GROUP−BY clause is issued and the colored parts of this query are extracted. Suppose that the group-by attribute used is a_3,where $a_3 = g_1$. The predicate values (x_1, x_2, x_3, x_4) and the extracted group values $(G_1, \ldots, G_k)$ are used to construct k meta-vectors in which the values for $(a_{1,lb}, a_{1,ub}, a_{2,lb}, a_{2,ub})$ contain the same values for all rows as the filter-predicates are applied to each group value G_i. The last two columns $a_{3,lb}, a_{3,ub}$ are used to store the group values. Each one of those meta-vectors will become associated with the output of the corresponding AF for the specific group. This is similar to the formulation of Database Learning

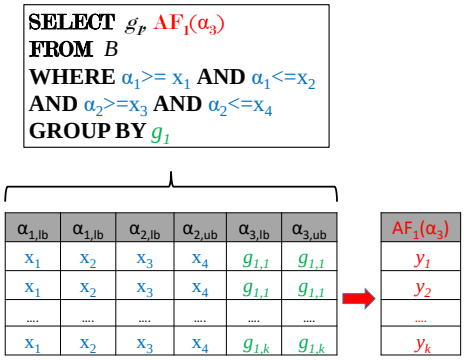

Figure 2.3: How to vectorize GROUP−BY queries.

[99]. However, in [99], the number of distinct groups is limited to 1000, as the individual AFs are executed on-the-fly. In ML-AQP there is no restriction, hence, an arbitrarily large number of groups can be supported by the formulation.

2.3.2 Handling Categorical Attributes

Some attributes might hold categorical values instead of numerical. An accepted approach is to restrict the length of the categorical attributes to the currently longest of each categorical attribute [77]. Subsequently, ASCII codes can be obtained for the remaining characters, which are concatenated to represent the value of G_i as in [77]. One other option is to construct various *dummy* columns each one denoting a value included in the categorical attribute a_i[80]. Suppose N distinct values for $a_1 = (A_1, \ldots, A_N)$, then N dummy columns are created, with its rows having a value of $\{0, 1\}$, thus producing the mapping $A_i \rightarrow \{0, 1\}^N$. However, an inherent problem with this option is the dimensiolity increase of query vector $\mathbf{m}$ as its dimensionalty becomes $2d + N$. Especially for large number of distinct values, N, this approach becomes extremely inefficient. To this end, an effective encoding scheme can be an injective function, such as various hash functions, that provide an effective mapping from a categorical attribute to a real number, i.e., $\mathcal{A} \mapsto \mathbb{R}$. In the implementation of ML-AQP we use a combination of both. The first technique is used for attributes with low cardinality

< 1000. For tree-based ML algorithms, this has been shown to work best [80]. The latter technique is used for attributes with high cardinality > 1000.

Empirically, we have found both of these approaches to work well. However, we also employ an additional optimization to increase accuracy for `GROUP-BY` queries with large number of distinct categorical values of the grouping attribute. We make use of an encoding scheme which effectively maps each categorical value to a real number as its the default behavior of ML-AQP. However, we also incorporate information in the encoding such that it becomes correlated with the output and can assist in increasing the accuracy of ML models. Let N distinct values for grouping attribute $a_1 = (A_1, \ldots, A_N)$, then $l(\cdot)$ is an encoding function, mapping categorical values to natural numbers $l : A_i \rightarrow \mathbb{N}$. Now suppose that we have a number of previously executed queries that include a `GROUP-BY` clause with the grouping attribute a_1. We can obtain the mean response of each one of those groups $(\overline{y}_1, \ldots, \overline{y}_N)$, where the responses for n queries per group $\overline{y}_1 = \frac{1}{n}\sum_i^n y_{1,i}$, are used to obtain the mean responses. Then we incorporate this information into our encoding scheme as shown by (2.1) :

$$h(A_i) = log(l(A_i)) + \overline{y}_i \tag{2.1}$$

Eq. (2.1), maps categorical values to numerical values that are largely determined by the mean response of their associated group while also incorporating a small offset to account for groups with similar mean response values but different categorical values.

2.3.3 Overall Support for Queries & Limitations

Overall, with the proposed representation we are able to support a large fraction of the aggregate queries commonly in an Online Analytical Processing (OLAP) setting. However, we can also support more modern types of analytics queries that are commonly found in deployments processing different data types, such as Internet of Things (IoT) or Spatial Data and Maps. These data types include spatial/temporal components which the users restrict, to focus their attention on particular areas. These type of queries can be modeled as range queries, restricting the temporal/spatial dimensions and extracting statistics using AFs. Therefore, ML-AQP is not restricted to a basic OLAP setting, on the contrary, it can provide support for a wide range of deployments. We also empirically verify this in the Experimental Section of this chapter in section 2.6.5. We can support simple multi-predicate aggregation queries to queries that include JOINs and GROUP-BYs. We can provide support for foreign-key joins as this is the case for multiple AQP engines [99]. Specifically, our solution does not make any assumptions as to what type of AFs are used. To ML-AQP, the response variable is a scalar y, associated with a meta-vector $\mathbf{m}$. Subsequently, it tries to identify patterns in $\mathbf{m}$ that would allow it to predict a future y_{new} when given an $\mathbf{m}_{\text{new}}$. Therefore, it is agnostic to the AFs used. This is in contrast to most sampling-based AQP engines [12] which restrict

the number of supported AFs to `COUNT`,`SUM` and `AVG`. In addition, in the presence of textual filters,(`LIKE '%product'`), ML-AQP leverages the approach outlined in section 2.3.2.

For JOINs which do not simply extend the dimensionality, but instead introduce less/more tuples in the result, we do not explicitly represent them in the current meta-vector. As we described, usually such schema designs are avoided when conducting analyses over large amounts of data [59]. In addition, derived attributes for GROUP-BYs cannot be supported with our current formulation and instead such queries have to be partially executed to obtain the derived attributes.

A key limitation is that we do not explicitly account for different logic operators ($\neg, \vee, \wedge$). We discuss a possible approach to address this problem even though we do not currently incorporate it in the implementation of ML-AQP. Given meta-vector $\mathbf{m} \in \mathbb{R}^{2d}$ this vector can be augmented with d more elements $\mathbf{o} \in \{-1, 0, 1, \emptyset\}^d$ which denote the existence of different logic operators. Hence, differentiating on cases where predicate values are the same but joined by different logic operators.

In addition, when handling GROUP-BY queries we rely on a `DISTINCT` operator to extract the different groups for the corresponding attributes. However, this approach does not account for a grouping attribute that is included in the `WHERE` clause. This could result in filtering out some of the groups a-priori. This is mitigated by applying the filter after retrieving the values of the grouping attribute.

2.3.4 Restricting Dimensionality of the Query Representation

As the number of columns (attributes, dimensions) gets larger, the representation will be moving towards a high-dimensional space causing problems to underlying ML models. One way to tackle this is to use unsupervised dimensionality reduction techniques [107], which will reduce the dimensionality of the given query vectors. Another, more straightforward way to tackle high-dimensional meta-vectors, is to restrict the number of columns that are used. Prior work has examined multiple query workloads [12, 137] and has shown that these queries usually focus on a subset of the original column set. This suggests that the meta-vector could be significantly smaller and only use the columns that appear frequently in the query logs. Especially, in the case of spatio-temporal datasets the focus is usually on the spatial and temporal columns to which an analyst applies filters and then examines descriptive statistics over other columns. Hence, meta-vectors can be constructed that only include these columns.

2.3.5 Aggregate Estimation and ML Models

Once a vectorized representation for queries is constructed, supervised regression algorithms are used to associate the meta-vectors with their results. An aggregate result is a scalar value $y \in \mathbb{R}$, thus, a dataset $\mathcal{C} = (\mathbf{m}_i, y_i)_{i=1}^{n}$ is derived from past query executions and their results. The goal of any statistical learning algorithm will then be to *minimize* the expected loss (difference or discrepancy) between the *true* query result of y and an estimated value of it, $\hat{y}$, derived from the trained ML model. In other words, the task is to approximate the conditional distribution $p(y|\mathbf{m})$ to minimize this loss. As each AF has a different underlying conditional distribution, different ML models are trained for each AF. Concretely, AFs such as `COUNT`, `SUM`, `AVG` will each be associated with specific ML model(s). Additional ML models are created for those AFs that refer to specific attributes. Although the number of ML models seems to be increasing, their storage footprint, as is examined in the experimental section, is minimal compared to the storage requirements of a sampling-based approach. It is important to note that models are not constructed a-priori, for each possible AF or combination of AF-attribute. Instead, a model is trained for an AF, if and only if there are queries in the query log referring to this specific AF. Implicitly, ML-AQP builds models over AFs that are frequently used. This significantly reduces the number of models that have to be built.

2.4 System Architecture

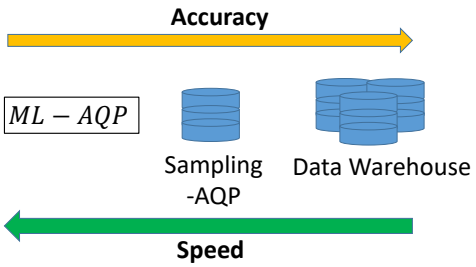

Figure 2.4: The ML-AQP within the complete data analytics stack. Starting from ML-AQP, analysts can choose a system going from left to right, if they require more accuracy. If speed is essential, they can choose from right to left.

Figure 2.4 shows holistically how ML-AQP complements the data analytics stack. Our system sits between cloud-based data warehouses and *sampling*-based AQP engines. Common cloud-based data warehouses, include Google's BigQuery and Amazon Redshift [57]. Using all three components (ML-AQP, sampling-based AQP engines, data warehouses), the analyst can choose which one to use, based on their needs of efficiency and accuracy. A system could also make this choice based on the resources available. Hence, if a Cloud system is

experiencing heavy loads, it could direct queries to either the sampling-based AQP (S-AQP) engine or ML-AQP. A useful analogy is to think of each of the three components as the *cache*, *RAM* and *Disk* components of a computer. Caches and RAMs, often, cannot hold the data required resulting in cache misses (hence the lack of accuracy), but the disk will always have the complete data set (will always have the true answer). However, fetching data from disk comes at a cost in efficiency. Therefore in our case, ML-AQP can act as the cache of the data analytics stack, sampling-based AQP engines as the RAM and finally cloud-based engines as the disk.

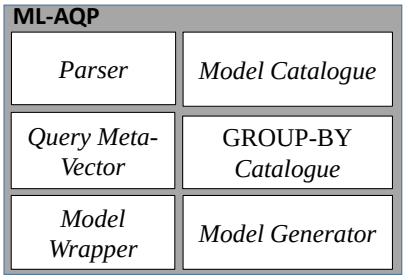

Figure 2.5: ML-AQP system architecture

To better explain the overall architecture followed in ML-AQP, we explain the role of each component within two distinct modes: (a) *Training* mode and (b) *Prediction* mode. During *Training* mode, queries are either executed at the Data Warehouse or the S-AQP and become associated with their results. We can also utilise pre-computed queries stored in log files. ML-AQP leverages those queries to build training sets of query-result pairs for the ML models. Training ML models transitions ML-AQP to the *Prediction* mode in which queries are transformed into the described vectorial representation $\mathbf{m}$ and their results are estimated by ML models.

All individual components that make ML-AQP are shown in Figure 2.5. The complete flow and interaction of the components, is shown in Figure 2.6. Initially, at *Training* mode, each query is parsed, through the *Parser*, and the projected AFs are extracted $f_1, \ldots, f_n$, along with the included predicates $p_1, \ldots, p_m$ and any `GROUP-BY` attributes $g_1, \ldots, g_k$.

In the example in Figure 2.6, the extracted AF is $f_1 = \texttt{AF}_1(a_3)$, the resulting predicates are $p_1 = (a_1 \geq x_1)$ and $p_2 = (a_1 \leq x_2)$ and the `GROUP-BY` attribute is g_1. For the predicates, we construct an $\mathbf{m} \in \mathbb{R}^{2d}$ meta-vector, where d is the total number of attributes in the data set. Each meta-vector is associated with a number of results $y_1, \ldots, y_n$, obtained from the executed AFs. In this example the meta-vectors are associated with a single result y because a single AF is used.

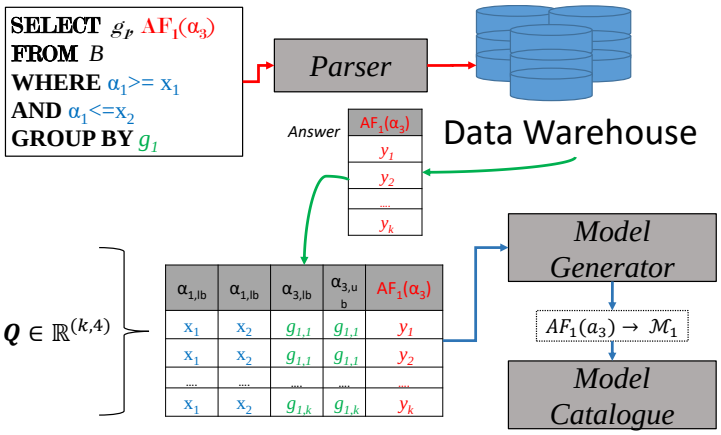

Figure 2.6: ML-AQP during Training

If no `GROUP-BY` clause is used, then we call this a single query $\mathbf{q} = (\mathbf{m}, \mathbf{y}), \mathbf{q} \in \mathbb{R}^{2d+n}$. For any `GROUP-BY` attribute g_i, a `SELECT-DISTINCT` query is executed, for attribute g_i, and its result is cached in the `GROUP-BY` catalogue D. The catalogue D is a mapping from the `GROUP-BY` attribute g_i to its set of distinct values $D : g_i \rightarrow \mathcal{G}_{g_i}$. Caching its result allows their reuse during *Prediction* mode. Given the values returned for g_i, $\mathcal{G}_{g_i} = \{G_{g_i,1}, \ldots, G_{g_i,|\mathcal{G}_{g_i}|}\}$, we construct multiple single-queries which have different results, $y_1, \ldots, y_k$ as they correspond to different groups. Hence, in the case of `GROUP-BY` queries, a single query has a matrix representation holding its meta-vectors and their associated results $\mathbf{Q} = (\mathbf{M}, \mathbf{Y}), \mathbf{Q} \in \mathbb{R}^{(|\mathcal{G}_{g_{(i)}}|)\times(2d+n)}$, also depicted visually in Figure 2.6.

The same procedure occurs for every executed query. This results in the collection of possibly sparse vectors, as in a typical query, not all attributes in a schema are included in its predicates. Because of this, we store all of the processed queries in a sparse matrix to reduce storage overhead. Once we finish parsing and constructing the representation for each query in our training set, we use the *Model Generator* to construct/train models $\mathcal{M}_i, \ldots, \mathcal{M}_n$ and associate each model with a specific AF. The models are then serialized and stored in the *Model Catalogue*.

A similar process occurs in *Prediction* mode when a new (SQL) query is issued to ML-AQP. The complete flow of interactions between the components is shown in Figure 2.7. During initialization of ML-AQP, both the `GROUP-BY` catalogue and *Model Catalogue* are loaded in memory. Consider the same example query shown in Figure 2.6. But, at this point the query is not executed, instead, ML-AQP predicts its answer. Specifically, the *Parser* is used to extract the same elements (predicates, AFs, `GROUP-BY` attributes). A vectorized representation of the query is constructed using *Query Meta-Vector*. If a `GROUP-BY` statement exists, the resulting meta-vector is a matrix $\mathbf{M}$ and the values for $g_i, \ldots, g_k$ are obtained from the `GROUP-BY` catalogue storing different $\mathcal{G}_{g_i}$. Next, all necessary AFs, to be estimated, are identified and their models are fetched from *Model Catalogue*. The *Model Wrapper* is used

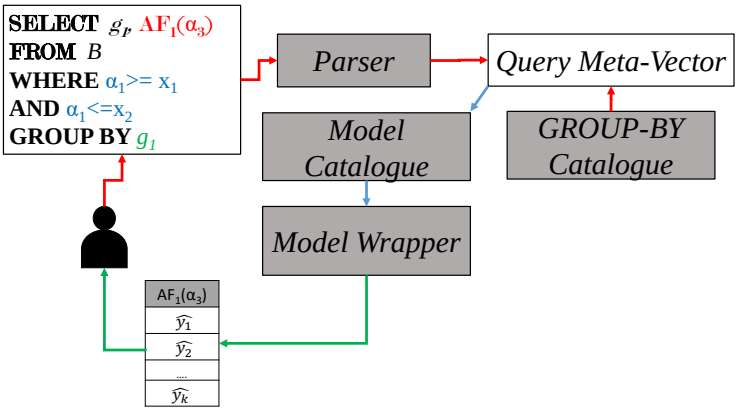

Figure 2.7: ML-AQP in Prediction mode

to query the model and estimate results given meta-vector $\mathbf{m}$. The result(s) are then returned to the user in an efficient manner as no data are accessed and the only overhead is the inference time of a model.

2.5 Machine Learning Specifics

Each query result y from training pairs $(\mathbf{m}, y)$, is derived from an unknown truth function $f(\cdot)$. Such function produces answers with respect to an unknown conditional distribution $p(y|\mathbf{m})$. Our aim, is to approximate the *true* function f for each aggregate function e.g., `COUNT`, `AVG`, `MIN`, `MAX`, `SUM`, etc. Supervised regression models are suitable for this task.

Initially, a set of queries and their responses $\mathcal{C} = \{(\mathbf{m}, \mathbf{y})\}, \mathbf{m} \in \mathbb{R}^{2d}, \mathbf{y} \in \mathbb{R}^n$ is obtained. Queries that have a matrix representation $\mathbf{Q}$, are treated as collections of single queries. Essentially, each row in $\mathbf{Q}$ corresponds to a single query as the `GROUP-BY` attribute can be reconstructed into a single predicate restricting attribute g_i to a single value. The task is to train ML models $\mathcal{M}_i, \ldots, \mathcal{M}_n$ that produce regression functions $\hat{f}_i, \ldots, \hat{f}_n$ that minimize EPE. Multiple such regression algorithms exist and to make the right choice we have to consider some of the properties of the problem at hand.

2.5.1 Choice of Machine Learning Models

A primary concern is that the produced training set $\mathcal{C}$, is inherently sparse as both the parameter vector $\mathbf{m}$ and response vector $\mathbf{y}$ contain `NULL`s (which can be represented as zeroes in linear algebra). This happens in cases where: (a) attributes do not have any predicates imposed on them and, hence, $a_{i,lb}, a_{i,ub}$ would be `NULL` or zero[3] (b) different queries might

[3]A special construct *nan* is placed instead of 0 as 0's are perfectly acceptable values.

use different AFs. If a query does not use all encountered AFs in set $\mathcal{C}$, then the response for the excluded AFs is set to `NULL`. For instance, in a dense matrix representation consider two queries calling different AFs; $\texttt{AF}_1$ and $\texttt{AF}_2$. To represent those queries in a matrix we have to set up two columns, $\texttt{AF}_1$ and $\texttt{AF}_2$ with the first row (for the first query) having a `NULL` value for $\texttt{AF}_2$ and the second row having `NULL` for $\texttt{AF}_1$. However, `NULL` response values is problematic as they are considered as undefined and are subsequently dropped during training of the ML models. To alleviate this problem, we partition the dataset per AF, such that queries that refer to the same AF are grouped together. This ensures that the response column does not have undefined values. However, the input matrix $\mathbf{Q}$ is still sparse as we have addressed case (b) but not case (a). Hence, we need algorithms that are able to handle sparse input.

In addition, we might have to deal with a large number of training queries, so we need algorithms that are scalable. For instance, we exclude the use of Support Vector Regression [125] as we have found that they cannot handle a large number of training examples if the implementation solves the dual problem in its closed form.

Linear models can often be trained in an online manner using SGD [30], which makes them very efficient. However, they result in simple models which cannot adequately model non-linear relationships without introducing more terms. Models such as Ridge and Lasso [46] regression are interesting variants that include regularization to handle the increased dimensionality of our input. However, we have found that these algorithms do not perform well to our problem. We have also considered the use of Deep Learning, however the models become unnecessarily complex, expensive to train, difficult to tune and have high inference times [77] which could increase the latency of estimating the response of an issued query.

In light of all these, we have made the choice of using Gradient Boosting Machines (GBM) [49] using efficient-parallel implementations called `XGBoost`[35] and `LightGBM`[71]. A GBM iteratively fits decision trees at first trying to approximate the response variable and then making this approximation more fine grained by combining decision trees trained on the negative gradient of the response variable and the produced predictions by the last decision tree. In addition, the highly scalable implementation of GBM by XGBoost and LightGBM allows handling large, high dimensional and sparse input.

2.5.2 Error Guarantees

A necessary feature of any AQP engine, is its ability to provide probabilistic error guarantees, for each predicted response. If an AQP engine provides an estimate, $\hat{y}$, for an AF, then the user has to know whether the *true* value y would be within an interval. This interval is associated with a certain probabilistic guarantee. Most AQP engines provide these guarantees by constructing confidence intervals [98, 70, 12, 95].

However, as ML-AQP is not sampling-based and, instead, relies on ML models, no confidence intervals can be built on its estimates/predictions. Confidence intervals are produced because it is assumed that the produced $\hat{y}$ (from sampling-based AQP engines) is an estimator of a population parameter. However, in ML-AQP, $\hat{y}$ is a prediction given some information on query vector $\mathbf{m}$. In this section, a naive solution of offering some kind of *error estimation* for aggregate answers is initially presented. However, more promising approaches are then discussed with the method of choice analysed in the end of this section.

All ML models try to minimize the EPE which was initially defined in (1.1). During *Training* mode, EPE is an over-optimistic estimate of the *generalizability error* that a specific ML model is associated with. The generalizability error is an estimate of the error associated with any future estimations. However, because ML models are trained on the set used for measuring EPE, they tend to produce inaccurate estimates for EPE. Hence, the use of cross-validation [46] is often employed. Cross-validation, measures the EPE on *out-of-sample* examples that the model did not use during training. Although, techniques such as Leave-One-Out (LOO) and K-Fold [46] produce good estimates for the EPE, the estimate is not probabilistically guaranteed. In addition, the EPE is *static* across the input space. Meaning that, even though an ML model might have learned to predict the answers of certain queries with error ϵ_1 and some others with error ϵ_2, and $\epsilon_1 \ll \epsilon_2$, both sets of queries will have the same EPE. We find the assumption of a static EPE across input space to be invalid and explore other suitable methods.

Instead of the estimate for EPE, we can use *Prediction Intervals*: Unlike confidence intervals, which are used to provide an interval for a population parameter, prediction intervals are used to provide intervals that contain the $true$ ($not\ predicted$) value of an aggregate result y_{n+1} of a future query (vector $\mathbf{m_{n+1}}$)). If we knew that the distribution of y is Normal and that any y is independent, prediction intervals could be produced similarly to confidence intervals. Using the sample of y given from the training examples $(y_1, \ldots, y_n)$, we compute the interval: $\overline{y} \pm t_a s_n \sqrt{1 + \frac{1}{n}}$, where t_a is the $100(1 - \frac{a}{2})^{\text{th}}$ percentile of the t-distribution, with $\alpha \in (0, 1)$ and commonly set to $\alpha = 0.01$ or $\alpha = 0.05$, s_n is the sample variance of the response variable y and $1 - \alpha$ is the *coverage* of the prediction interval. However, we do not wish to make any parametric assumption about the distribution of y. Hence, we resort to other methods outlined below.

A prominent method is *bootstrap* [42], which makes no parametric assumption as to the distribution of y. This is a common approach encountered in sampling-based AQP engines [146, 11]. In sampling-based AQP engines, the *bootstrap* method is used and the underlying dataset is re-sampled b times (where b is usually over > 100) to produce a distribution of estimates for y, $\hat{y}_{i,1}, \ldots, \hat{y}_{i,b}$. Let $y_{i,0}$ be the original estimate, with the estimates provided by the *bootstrap* samples, the residuals $y_{i,0} - \hat{y}_{i,1}, \ldots, y_{i,0} - \hat{y}_{i,b}$ are computed. Using the empirical distribution of residuals, quantiles can be computed. The quantiles can be used

to produce a confidence interval $[\hat{y}_{i,0} - t_{1-a/2}, \hat{y}_{i,0} + t_{a/2}]$. Theoretically, ML-AQP could also use the *bootstrap* method, re-sampling the training dataset b times and constructing b ML models $M_i, \ldots, M_b$. This would yield b estimates for y, $\{\hat{y}\}_{i=1}^{b} = \{\hat{f}_i(\mathbf{m})\}_{i=1}^{b}$ and can similarly produce confidence intervals. However, this methodology would incur the costs of training, maintaining, and predicting the estimates from $b+1$ (if we count the initial model providing the prediction) different ML models. Multiply that by the number of different AFs that need to be learned and the overhead cost of this approach quickly becomes huge.

More recent developments in ML literature focus on building predictive intervals by using *conformal inference* [122, 82, 97]. This technique relies on building a *non-conformity* measure which estimates the difference of two examples i.e., $\mathbf{m}_i$ and $\mathbf{m}_j$. This could be defined as the L_p norm (i.e $p = 2$) of the examples $\|\mathbf{m}_i - \mathbf{m}_j\|_2^2$. But finding the right non-conformity measure in our case is non-trivial as the input vectors are high-dimensional and sparse. Distance in this case becomes meaningless [13] and the choice of a *valid* p-norm is beyond the scope of this work. In addition, these techniques scan the complete set of previous training examples to find similar and dissimilar examples. Meaning that all previous queries have to remain stored. This is not ideal, as the set of all queries would have to be deployed to every location that ML-AQP is served.

Therefore, the most favourable choice, when designing ML-AQP, was Quantile Regression (QR) [76]. QR offers an alternative method of providing prediction intervals to estimates that: (a) does not require the storage of training examples (b) does not have the overhead of training/maintaining a large number of additional models and (c) does not make parametric assumptions to the distribution of the response variable y.

Typical regression models minimize the EPE and focus on estimating the conditional expected value of y, $\mathbb{E}[y|\mathbf{m}]$. QR estimates the t^{th} conditional quantile $Q_{y|\mathbf{m}}(t)$. Multiple ML algorithms have been proposed to estimate conditional quantiles [126, 90, 129]. Formally, given a conditional distribution function for y,

$$F_{y|\mathbf{m}} = \mathbb{P}\{Y \leq y|\mathbf{m}\}, \tag{2.2}$$

we define the t^{th} conditional quantile function as:

$$q_t(\mathbf{m}) = \inf\{y \in \mathbb{R} : F_{y|\mathbf{m}} > t\} \tag{2.3}$$

Where $\inf$ is the *infimum*, which points to y that is less than or equal to all the elements in the defined set. Given the conditional quantile function defined in (2.3), we construct prediction intervals using $[q_{t/2}, q_{1-t/2}]$. This defines the lower and upper bounds of the estimated value for y with coverage probability of $100(1-t)\%$. As stated earlier regression algorithms estimate the conditional expectation of y, $\mathbb{E}[y|\mathbf{m}]$ by minimizing EPE. In the same manner,

quantile regression algorithms estimate the conditional t^{th} quantile $Q_{y|\mathbf{m}}(t)$ by minimizing what is known as "*pinball loss*"[76] :

$$\rho_t(y, \hat{y}) = \begin{cases} t(y - \hat{y}), & \text{if} \quad y - \hat{y} > 0 \\ (1 - t)(\hat{y} - y), & \text{otherwise} \end{cases} \tag{2.4}$$

Supposing we have trained two different quantile regression functions $\hat{q}_{t/2} : \mathbf{m} \in \mathbb{R}^{2d} \rightarrow Q_{y|\mathbf{m}}(t/2) \in \mathbb{R}$ and $\hat{q}_{1-t/2} : \mathbf{m} \in \mathbb{R}^{2d} \rightarrow Q_{y|\mathbf{m}}(1 - t/2) \in \mathbb{R}$. Then, a prediction interval for each new query is estimated as: $[\hat{q}_{t/2}(\mathbf{m}_{\text{new}}), \hat{q}_{1-t/2}(\mathbf{m}_{\text{new}})]$ with coverage probability $100(1 - t)\%$.

Therefore, ML-AQP provides error guarantees using QR and the statistical tools of prediction intervals and coverage. Specifically, ML-AQP produces a prediction interval $[low, high]$ and a coverage level $l\%$ and guarantees that the $true$ answer to a future query will fall within $[low, high]$ with probability $l\%$. LightGBM [71] offers support for QR and we are also looking into incorporating [106] to support stricter guarantees.

2.6 Evaluation

2.6.1 Experimental Setup

Datasets & Workloads. For our experiments we used the following data sets and workloads[4]:

1. `TPC-H`[5]: This is the standard TPC-H benchmark.
2. `Instacart`[2]: This is a data set of an online store. A database was created using the csv files which follows the setup of VerdictDB. [98].
3. `Crimes`[6]: This is a real data set of crimes reported in the city of Chicago. A workload for this data set was obtained from [1] which models a number of range-queries with multiple AFs. Their predicates are sampled from a number of random distributions to simulate various analysts executing queries at different subspaces of the data set.
4. `Sensors`[8]: Is a data set comprised of a number of sensor readings including voltage, humidity temperature etc with a temporal dimension. A synthetic workload was created restricting the temporal dimension and extracting the `MIN(temperature)` and `MAX(humidity)`.

[4]All experiments can be found at : `https://github.com/Skeftical/modelbasedaqp`

5. `synthetic`: A synthetic workload was constructed to stress test our chosen representation. The set of workloads include up to 100 attributes (stretching our meta-vector to 200, $\mathbf{m} \in \mathbb{R}^{200}$) with the number of set predicates up to 50.

Training ML-AQP. ML-AQP has a Training phase, much like sampling-based AQP engines that need to create samples before being able to operate. For all workloads, we generate queries and train the models on 70% of the complete workload. We then conduct the experiments and take measurements on the rest 30%. This is standard practise in the ML literature. It is a cross-validation strategy, much like the well-known K-Fold validation, where the generalizability error of the model is estimated by a single test-set. This strategy can be a good approximation to the true generalizability error in cases where the number of training samples is large [46]. For `Instacart`, we use a similar format of queries as VerdictDB [3]. However, to facilitate learning we vary the predicate values. For all queries containing predicate values, we generate queries from the same template with values sampled from a Normal distribution $\mathcal{N}(\mu, \sigma)$, where (μ, σ) is the average and standard deviation of the corresponding attribute, respectively. If the attribute contains a categorical value, the generated queries contain a value selected uniformly at random. The total number of training queries generated were 10^4 and $3.3 \cdot 10^3$ for testing. Some queries contained no predicates, in these cases no additional queries were generated. The number of queries obtained is not large as typically there are millions of queries being executed on a daily basis in production environments [70].

For `TPC-H`, we use a subset of the queries contained in the benchmark as we are still making progress on our *Parser*. We generate 100 queries for each of the queries used. A model is trained for each distinct AF. For `Instacart`, three different models are generated as three distinct AFs are used in this workload. For `TPC-H`, 12 different models were generated. For `Crimes` and `Sensors`. we generate a model per AF tested.

2.6.2 Efficiency

We first examine the efficiency of ML-AQP and demonstrate the speedup gains over a popular database, PostgreSQL. We compare the results using a sampling based AQP engine, VerdictDB [98]. Let t_b be the response time for PostgreSQL and t_m, t_v the response times for ML-AQP and VerdictDB, speedup is measured by $\frac{t_b}{t_m}$ and $\frac{t_b}{t_v}$, respectively. For this experiment, we use `TPC-H` with 1GB and `Instacart` with its main fact tables (`order_products`, `orders`) containing 3M and 30M rows. For `TPC-H`, we use a subset of all the template queries and for `Instacart`, we use the same format of queries as used in the evaluation of VerdictDB[3]. For VerdictDB, uniform samples were created for large fact tables at $1\%/10\%$ ratio. This experiment ran on a single machine with an Intel(R) Core(TM) i7-6700 CPU @3.40GHz, 16GB RAM and 1TB HDD.

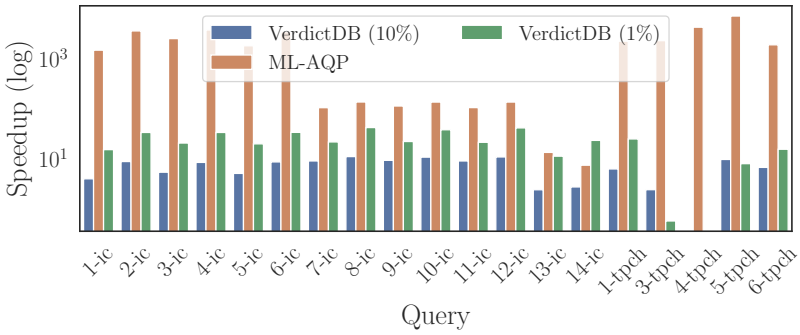

Figure 2.8: Speedups offered by ML-AQP compared to VerdictDB

System	**Time** (sec)	95^{th} percentile (sec)
PostgreSQL	1.62 ± 1.21	4.01
VerdictDB (10%)	0.28 ± 0.41	0.79
VerdictDB (1%)	0.14 ± 0.3	0.43
ML-AQP	0.05 ± 0.16	0.12

Table 2.1: Performance over all queries across systems

The results are shown in Figure 2.8. This figure demonstrates how much faster VerdictDB and ML-AQP are in terms of the response times given by PostgreSQL. We can instantly notice that the speedup differences are huge (notice the log-scale on y-axis)[5]. Even though we are using relatively small datasets, VerdictDB, understandably, cannot offer the same speedup as ML-AQP. The minimum/maximum speedup gained by ML-AQP is at $7\times$/$7200\times$,for VerdictDB 1% $0.59\times$/$43\times$ (as we suspect that some of the computation is offloaded to the main engine) and for VerdictDB 10% $3\times$/$11\times$. This stems from the fact that ML-AQP is only performing inferences at *Prediction* mode using trained models. It does not need to scan any of the data at any time. To be more specific, Table 2.1 shows the mean response time along with the standard deviation and 95^{th} percentile for all queries across the four different systems. As it is evident, even at the 95^{th} percentile the response times for ML-AQP are no greater than 120 milliseconds, satisfying the interactivity constraint set at 500ms [85].

Even for queries with relatively large `GROUP-BY`s the speedup is at $20x$. By default `GROUP-BY`s are a bottleneck in our case as multiple queries have to be executed for each distinct value of the attribute used in the `GROUP-BY` clause. For instance, query 14-ic has approximately 50K distinct values. In this experiment its values were cached as this would have been the default behavior. This is due to similar queries with the same GROUP-BY attributes being executed during *Training* mode. When caching the values, query $14 - ic$ takes 0.67 seconds to execute and 0.7 seconds when it does not cache the GROUP-BY values. We can see minor

[5]For 4-tpch we could not get VerdictDB to execute this query.

impact, with an overhead attributed to the execution of the `SELECT DISTINCT` query at 0.3 seconds. We can still get $7\times$ better response times than PostgreSQL, where as VerdictDB (1%/10%) is at $24\times$ /$3\times$ for this particular query. Although a larger speed-up is observed, for VerdictDB(1%) we will notice that accuracy using 1% sampling ratio deteriorates with large errors in the individual groups.

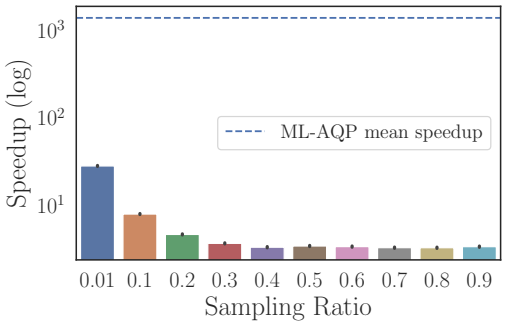

Figure 2.9: VerdictDB speedup for an increasing sampling ratio.

In Figure 2.9 we examine how the speed-up offered by sampling-based, and in general data-driven AQP, solutions diminishes as the sampling ratio is increased. For point of reference, we also provide the average speedup offered by ML-AQP which is not constrained by a sampling ratio as it uses no data.

2.6.3 Efficiency in the Cloud

Our solution is designed to alleviate the monetary, computational, storage costs in large deployments usually in the Cloud. We first examine how the computational cost can be mediated using our solution. For this experiment, we use *AWS Redshift*, with 2 `dc2.large` compute nodes and 1 master node each at 16GB memory with 160GB SSD. We use a $100\times$ scaled version of the `Instacart` dataset. The total storage footprint of this data set is 100GB with the main fact tables (`order_products`, `orders`) containing 4.2 billion and 0.5 billion rows respectively. We execute the same `Instacart` queries [3] and we set a `timeout` value at 5(mins). After this timeout period, we abort the execution of the query. In this experiment, we aim to show how data-driven methods are strictly coupled with data. An increase in data size unavoidably increases response times. This effect is not evident in ML-AQP. For VerdictDB we uniformly sample the same fact tables at 10% ratio. In this experiment, we expect the results for VerdictDB to deteriorate. On the other hand ML-AQP is constant in its performance as it is unaffected by data size. It is important to recall that the deployment of ML-AQP can happen in two ways: (i) All the models and required modules for ML-AQP can be distributed to all the analysts' machines and be loaded in memory during

ML-AQP (*local*)	ML-AQP (*network*)	VerdictDB
443-10^5	24-195	0.16-28

Table 2.2: Minimum-Maximum speedups at the Cloud

analysis (later experiments will showcase that the small storage footprint of ML-AQP permits this solution); (ii) All models can be deployed at a server and be used as a service. This would have significantly lower costs than executing queries using Redshift. However, it might have more overhead as the predictions have to be transferred to the analysts machine over the network. We further examine the performance benefits of both solutions.

The results of this experiment are shown in Figure 2.10. There are two different deployments for ML-AQP: (a) ML-AQP *(network)*, (b) ML-AQP *(local)*. For ML-AQP *(network)* deployment, we set up a small server serving predictions over the network. It accepts HTTP POST requests with the extracted parameter values of the SQL query and returns a prediction of its answer. The results shown in Figure 2.10 are in *log-scale*. As expected, the benefits of a *local* deployment are far greater, although we would have to consider problems in maintaining the models as in this case ML-AQP are in each analysts machine. In addition, for some queries VerdictDB offers no speedups as Redshift is able to process those queries in an efficient manner.

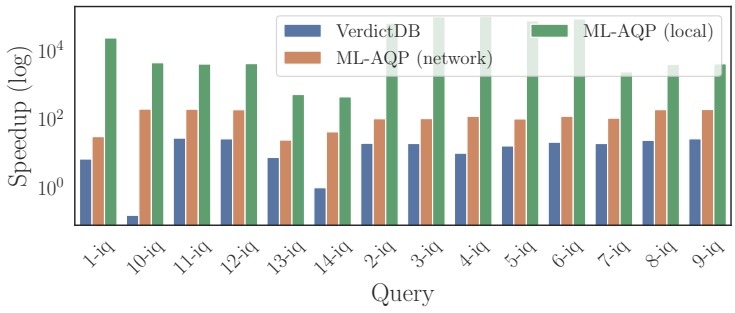

Figure 2.10: Speedups in large deployments

To be more concise, the min/max speedup benefits of the compared systems are shown at Table 2.2. As evident, the *local* deployment is orders of magnitude faster than both VerdictDB and an over the *network* deployment. We also report on average response times and the response times at the 95^{th} percentile for all systems. The results are shown in Table 2.3. The first thing we notice, is that although VerdictDB has less mean response time than Redshift, at the 95^{th} percentile it is slower, possibly due to overheads of VerdictDB in deciding which samples to process. In addition, both the *network* and *local* deployments for ML-AQP offer mean sub-second latencies and only 2.68 seconds at the 95^{th} for *network*.

System	**Time** (sec)	95^{th} percentile (sec)
Redshift	55 ± 75	142
VerdictDB	49 ± 120	300
ML-AQP (*network*)	0.78 ± 1.82	2.68
ML-AQP (*local*)	0.02 ± 0.08	0.25

Table 2.3: Performance in the Cloud

2.6.4 Training Overhead

As stated earlier, ML-AQP has to go through *Training* mode initially. At this stage, previously executed queries are used to train a variety of models and learn to predict the answers of future aggregate queries. Ideally, training the models would happen *locally* at Data Scientist's machines so as not to incur additional costs of repeatedly training and fine tuning the models in the Cloud. Therefore, in this experiment we measure the Training Time required to build a model with varying number of queries. We run this experiment locally on a single machine with an Intel(R) Core(TM) i7-6700 CPU @3.40GHz, 16GB RAM and 1TB HDD to demonstrate this capability. We compare this to the sample building time of VerdictDB with an increasing sample ratio. We use the `TPC-H` data set at 1GB. At each iteration 3 samples are built on the main fact tables. Figure 2.11 shows the result of this experiment.

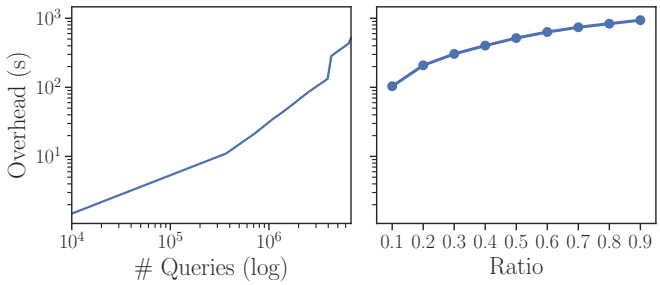

Figure 2.11: (Left) Training overhead for an increasing number of queries (x-axis) (Right) Sample preparation time for VerdictDB

The sample preparation time for VerdictDB, Figure 2.11(right), increases linearly and even for 1% ratio at 1GB, takes longer than training ML-AQP on 4 million queries. At $6+$ million queries, this overhead is still *less* than the sample preparation time for VerdictDB at 60% ratio. To put this in context, ML-AQP took 1.1 seconds, to train on $4,000$ queries generated for `Instacart`. For `TPC-H`, ML-AQP required less than a second to train each model. The only exception was for `COUNT` queries, executed over `Instacart`, as its associated queries included a relatively large number of groups $(> 50,000)$ and took 326 seconds to train on around 5 million training examples in total.

It is also important to note that sampling-based AQP engines are susceptible to the size of

the data set. In this experiment, we are only using 1GB of data, as the size increases, sample preparation time is expected to increase, too. This would not be a problem for ML-AQP as it is only affected by the number of queries and it is *not*, at any point, affected by the size of the underlying data set. In conclusion, both approaches, sampling-based AQP engines and QDL-based AQP engines will have "*training*" overheads. Their overheads are largely determined by different dimensions and as these solutions are designed to expedite query processing in *petabyte* scale storage engines we expect ML-AQPs overhead to be much less.

2.6.5 Accuracy

To assess the accuracy of ML-AQP we measure the *Relative Error*[98, 99, 12, 70] across all the query templates of both `Instacart` and `TPC-H`. ML-AQP was trained on past queries generated as described in Section 2.6.1. Three models were trained using `LightGBM` to answer `Instacart` queries, one for each AF involved. For `TPC-H`, 11 models were trained using `XGBoost` as the queries were largely referring to AFs on different attributes. The number of rounds were set to 10^4 with early stopping when no more improvement was shown. Objective was set to `squared_error`. We compare our results with VerdictDB, which created samples over the large fact tables at ratios of 1%/10%. An initial set of results is

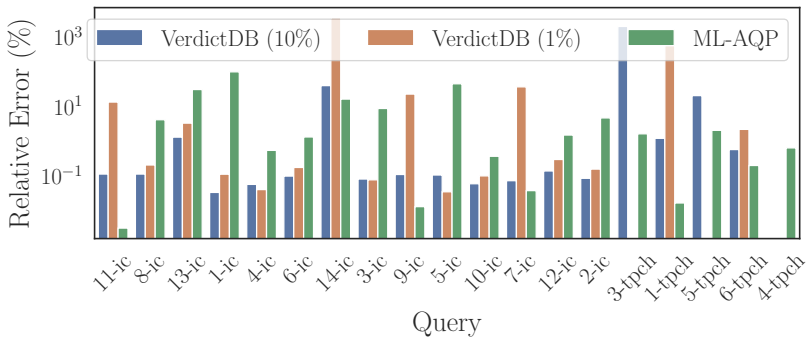

Figure 2.12: Relative Errors for each query in `TPC-H` & `Instacart`

shown in Figure 2.12. A first impression is that both systems perform really well over a large range of queries. Please note that the results show the average relative error over each query template. Where for each query template multiple, > 50, queries were executed with random predicate values as described in (2.6.1). ML-AQP is able to accurately answer 80% of queries for `Instacart` and 100% of the selected `TPC-H` queries with relative error below 10%. We can also visually discern that ML-AQP outperforms VerdictDB for many queries. VerdictDB at 1% was not able to answer accurately queries that have a large number of groups, such as 14-ic and in some cases the groups returned by VerdictDB did not match the

ones returned by the engine (3-tpch, 5-tpch)[6]. For ML-AQP, queries *1-ic* and *5-ic* produce large relative errors. This is understandable as these queries include no predicates and are simply the results over a full scan of the table (ie are simple `SELECT AF FROM T`). The results of such queries can easily be cached. ML-AQP is not expected to answer such queries as the meta-vector $\mathbf{m}$ is filled with `nan` values, to which the model simply ignores as there are no patterns to be learned.

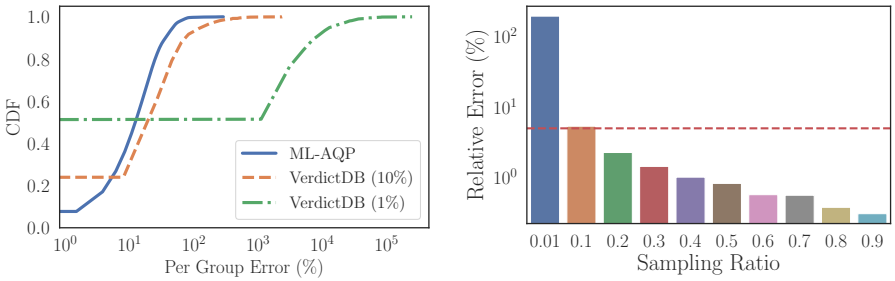

Figure 2.13: (Left) CDFs of relative error per group in a `GROUP-BY` query (right) Relative error for an increasing sampling ratio with the mean relative error for ML-AQP as a horizontal line.

We also measure the relative error on a per-group basis. Especially, for query 14-ic where a large number of groups are present we notice high relative error. Figure 2.13(left), shows the Cumulative Distribution Function (CDF) of the relative error across groups for all three systems. ML-AQP outperforms VerdictDB at both sampling ratios (1%, 10%), which shows that it can accurately estimate the aggregates across groups. Given the prior discussion we do not mean to say that sampling based engines have less accuracy. Instead, at a small sampling ratio the benefits are not great and the large trade-off between accuracy and speed makes their use inappropriate. Hence, sampling based engines can be used in parallel to ML-AQP, when the analyst needs more accurate answers and they are willing to sacrifice some of the efficiency for it, as also suggested by Figure 2.4. So, the systems can co-exist if we use sampling-based engines with a higher sampling ratio as the expected error over all queries decreases as is shown in Figure 2.13(right).

Accuracy on range queries over spatio-temporal data

We have also measured the accuracy of ML-AQP in predicting the responses of range-queries over spatio-temporal data sets. Specifically, multiple synthetic queries are executed over `Crimes` restricting its spatial dimensions and returning a response `COUNT`, `MEAN` or `SUM` over other attributes included in the data set. `COUNT`, returns the number of recorded incidents within the defined area, `MEAN` is the average *Beat* number which is a police defined

[6]We were not able to run query 4-tpch as an uknown error was thrown at runtime.

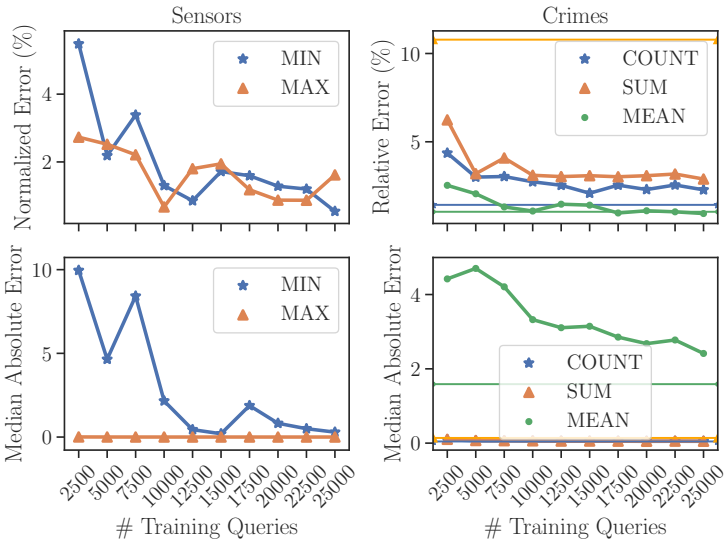

Figure 2.14: Accuracy over `Sensors` and `Crimes` for an increasing number of *training* queries and over different AFs. (Top) Relative/Normalized Error (Bottom) Median Absolute Error. (Right Column) For Crimes the accuracy of VerdictDB is plotted as horizontal lines.

number describing the area, and `SUM` of the arrests over the specified area. For `Sensors` we restricted the temporal dimensions and extract the `MIN`(temperature) and `MAX`(humidity). All of the results in Figure 2.14(top) show that the relative error is below the targeted 10% for this kind of data sets and conntinues dropping as more queries are being used for training the models. We also plot the accuracy for VerdictDB (1%) as horizontal lines only for the `Crimes` data set as VerdictDB does not support `MIN`/`MAX` aggregates. Note that for `Sensors`, we report on the Normalized Error $|\frac{y-\hat{y}}{\overline{y}}|$, which computes the absolute difference divided by the mean response. The reason is that for this workload, the values are really small ($\mu = 9$) and the measured relative error is not robust as it might report a 50% error even if $y = 2$ and $\hat{y} = 1$. This is also encountered in [69] and similar technique is employed. To provide more context as to how close the predictions are in relation to the true response, we also provide results on the Median Absolute Error (MAE) in Figure 2.14(bottom). It is a well known metric in the ML community that is robust to outliers indicating the median of the absolute error between y and $\hat{y}$. As evidenced, the absolute difference is small for all aggregates and data sets and continues to drop as more queries are used for training. The accuracy obtained is similar to VerdictDB's with ML-AQP having lower relative error for `COUNT`. In addition, we stress the fact that we are able to predict the responses for `MIN` and `MAX` that to our knowledge are not supported by most AQP systems. In addition, as the number of queries increase, we see a drop in relative error suggesting that more accurate predictions can be obtained. Overall, the results of this experiment show that ML-AQP is able to support a wide variety of aggregates over a diverse set of data sets.

Accuracy on error estimation

We study the effectiveness of the prediction intervals constructed using QR. For this experiment we train two `LightGBM` models on `quantile loss`, with parameters `n_estimators`= 1500 and `l_rate`= 0.001. We set $t = 0.95$ and train the two models using `alpha`= $1 - t$ and `alpha`= t. This effectively creates a prediction interval that would ideally provide a coverage rate of 90%. Coverage rate is used in other work to assess prediction intervals [82, 45] and is essentially an empirical estimate of the predictions that will fall within the proposed interval. It is computed using a held-out set of queries. Specifically, the two models generate responses for $\hat{y}_{5^{\text{th}}}$ and $\hat{y}_{95^{\text{th}}}$. We test each *true* value y on $\hat{y}_{5^{\text{th}}} \leq y \leq \hat{y}_{95^{\text{th}}}$ and report the ratio of queries where the condition is true. We used the queries of `Instacart` and conduct this experiment on three different AFs `COUNT`, `SUM`, `AVG`.

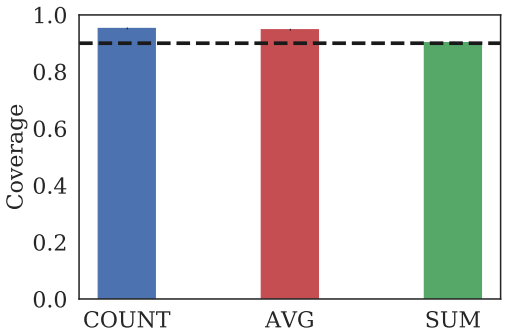

Figure 2.15: Coverage Ratio for different AFs. Horizontal line drawn at 90%

The results in Figure 2.15 confirm, that empirically a value lies within the interval, provided by QR estimates, by an estimated probability > 0.9. In short, using QR, ML-AQP is able to provide good probabilistic intervals for the true answer. Using this interval the user can choose whether they trust the prediction or they wish to get a more accurate estimate using an S-AQP or the data warehouse engine.

2.6.6 Storage

For this experiment, we measure the *Storage* overhead of ML-AQP. At the end of the Training phase we deploy ML models at analysts devices or a central device. Measuring the storage overhead and ensuring that this is adequately small is of great importance. We expect orders of magnitude smaller storage footprint than sampling based AQP engines as we neither store any of the data nor any of the queries used for training. We initially examine how much memory is required by a model with increased complexity. The main factor contributing to the size of the selected ML models (GBMs) is the number of trees and their depth.

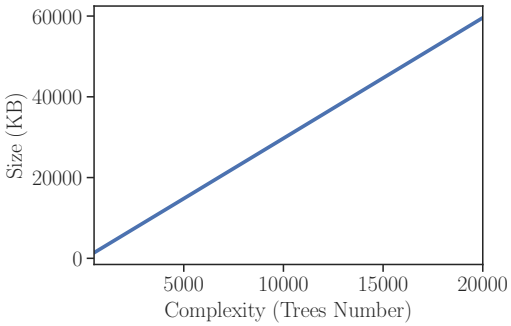

Figure 2.16: Increasing number of trees (x-axis) and size in kilobytes

Figure 2.16 shows an increase in the total storage required by an increasing number of trees. For conducting the experiments over `Instacart` and `TPC-H` the number of trees never exceeded 1700 with some AFs requiring as little as 100 trees. ML-AQP requires additional storage for encoding categorical values and for caching values obtained from queries with `GROUP-BY` clauses. For instance for `Instacart`, there are $50,000$ categorical values and ML-AQP requires an extra 4MB (on top of the storage required by the models). This cost increases linearly as the number of labels increase. Accounting for all of this and even any required modules by the implementation of ML-AQP still does not match the storage overhead required by sampling-based AQP. To put this in context, `Instacart` requires 2.4GB of storage for its tables. To sample its main fact tables `orders` and `order_products` at 1%, VerdictDB required 1.8GB in total. On the other hand, ML-AQP requires a mere 15.5MB to cover the aggregate queries issued against `Instacart`, this includes all models and catalogues. Given this information, we can safely assert that ML-AQP is extremely light-weight and can easily reside in main memory during analysis.

2.6.7 Sensitivity Analysis

In this section of our experimental analysis, we study a variety of variables contributing to the accuracy of our solution. For these experiments, we use the `synthetic` dataset to control the number of attributes and predicates set. Queries with meta-vectors up to 200, $\mathbf{m} \in \mathbb{R}^{200}$, are executed over uniform spaces with the number of set predicates up to 50. All predicates are numerical and each query vector is associated with a response. The predicates essentially define range queries over the respective columns. To put this in perspective, real workloads expect a median number of columns selected in a query around 8 [70] with a more recent estimate reporting that 90% of queries use around $1-6$ [69] columns with a maximum reaching 12. We increase the number of predicates and columns to study the effects on accuracy. We initially train the models on a constant number of queries $10,000$ and vary

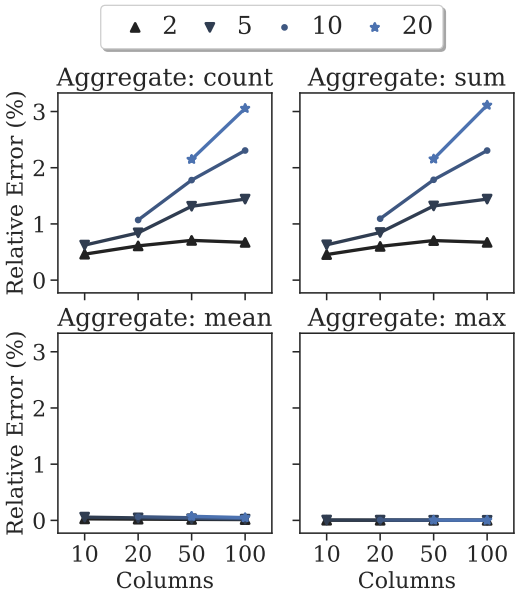

Figure 2.17: Relative Error (y-axis) measured across different aggregates with an increasing number of columns/attributes (x-axis) and a varying number of predicates set randomly.

the number of predicates and columns/attributes. In addition, we test different aggregates, `COUNT`, `SUM`, `MEAN`/`AVG` and `MAX`, to examine their predictability. In essence, ML-AQP is agnostic to what kind of aggregate is being predicted, to ML-AQP an AF applied over an attribute is a response variable to which it tries to identify patterns that can help it minimize the loss $L(y, \hat{y})$.

As can be seen in Figure 2.17, the relative error increases with respect to the number of columns and predicates. Although there is not a notable increase (2%), we can attribute this to the fact that more queries might be needed to learn a more complex space. As the dimensionality of the space (number of columns) increases, the number of predicates increasingly restricts the sub-spaces defined by the queries. In addition, For `MEAN` and `MAX`, we do not observe large differences in relative error. Closely, examining the workloads we notice that the Coefficient of Variation (CoV), defined by the standard deviation to the mean ration: $\frac{\sigma}{\mu}$ is 0.08 for the response y of `MEAN` and 0.01 for the response `MAX`. Where the CoV shows the extent of the variability in relation to the mean. A CoV value closer to 1 indicates high variability. Therefore, ML-AQP might be able to learn their distributions with less queries.

Figure 2.18 shows how relative error decreases as the number of queries that a model is trained on increases. For all aggregates, we notice that the benefit of a larger number of queries, diminishes and nothing more can be learned. This provides an approximation to the number of queries needed to model the specific aggregates under this workload. It also denotes that after exceeding a certain number of queries, we should then focus on the complexity of our model to further decrease the relative error. In addition, for `MEAN` and `MAX`,

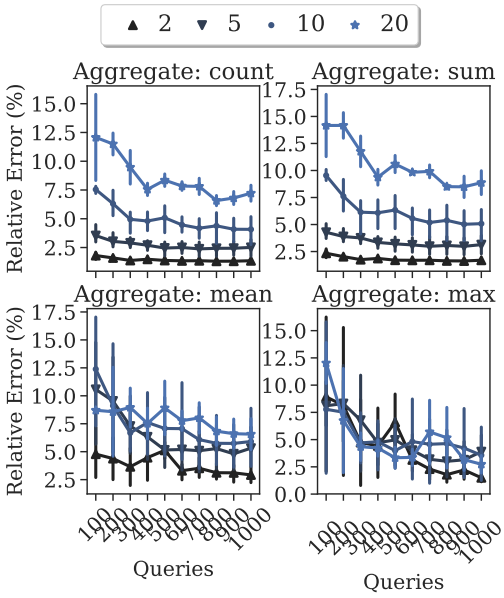

Figure 2.18: Relative Error (y-axis) measured across different aggregates with an increasing number of queries (x-axis) used for training.

we see high standard deviation for relative error across settings, especially when the number of queries is small. This advises us that as the number of queries is small, relative error is largely determined by the complexity of the workload (dimensionality, sparsity). However, as more and more queries are being used to train the models this effect fades away.

2.7 Conclusions

In this chapter we described ML-AQP, a QDL based, AQP system. ML-AQP offers a complementary approach to that of sampling-based AQP engines. The salient feature of ML-AQP is that it learns ML models over a set of previously executed queries instead of developing models or samples over (the potentially) massive base tables. The models are extremely compact and can predict the answers of future queries with small relative errors. Specifically, queries are transformed into a custom vectorized representation. Representing the queries and their answers as vectors allows training ML models that learn patterns to help them predict the response of future unseen queries. ML-AQP can offer orders of magnitude speedups in large deployments and he provided answers can be bounded by prediction intervals with high probability constructed by QR models. Moreover, a large fraction of aggregate queries, including `MIN`,`MAX` (which most AQP systems have struggled to address) can be supported along with `GROUP-BYs`. In general, there is no restriction to the aggregate function used. Our results show that ML-AQP can ensure low errors while introducing large efficiency gains with small memory/storage footprints, and supporting all aggregate functions. Finally, this shows how

QDL and ML in general can be incorporated into AQP without utilising any of the data.

Chapter 3

Query-Driven Explanations for Exploratory Analytics

3.1 Introduction

In this chapter we depart from the notion of incorporating QDL for AQP and show how this technique can be utilised to offer other kinds of functionality. In the era of big data, analysts wish to explore and understand data in an efficient and effective manner. The typical exploration procedure followed by analysts is rather ad-hoc and domain specific, but invariantly includes the fundamental step of *exploratory analysis* [66]. Exploring data spaces/regions is central for testing hypotheses, building predictive models, modeling data trends, etc.

To this end, AQs, e.g. queries that include aggregate functions such as `COUNT`, `SUM`, `AVG`, play a key role in exploratory analysis, as they *summarize* data regions of interest. The regions are often defined using query range operators. A range operator limits the number of returned rows (tuples/data items) by restricting the result set to rows within a given region. Using such summaries, analysts decide whether a data region is of high importance and whether they should continue exploring in this direction. In addition, the importance of AQs in data exploration is obvious as almost any operation can be described as a collection and/or combination of AQs. For instance, histogram construction can be achieved by executing a number of AQs using range queries. Extracting descriptive statistics for sub-spaces in a dataset, such as various moments (mean, variance, skewness, kurtosis) can also be described by such AQs.

However, AQs return scalars (single statistics/single values) conveying little information for further explaining the underlying data subspace defined by the retrieved tuples/data items. For instance, imagine determining whether a particular geographical region is of interest, depending on the number of persons, within this region, having relatively 'high' income. If

this number for a particular subspace is e.g., 273, then *what does this value mean and/or what other information one could extract by this statistic?* If the selected region, was less or more selective, then how would this statistic change? In addition, if the analyst wanted to identify the regions with the maximum/minimum statistic then a number of additional AQs would have to be executed. To answer such exploratory questions and enhance analysts' understanding of the queried subspace, one needs to issue more queries to further explore nearby regions.

In the beginning of the exploration process, the analyst has no holistic understanding of the data space, that would steer them in the right direction with respect to which and how many queries to issue next. Therefore, further exploration becomes ad-hoc, wandering, unsystematic, and uninformed. This might reflect additional cost (e.g., database access, additional computation and extra query processing cost) due to the execution of possibly redundant queries since no systematic guidance is provided to the analysts.

Our goal is to explain *how* an AQ result over a data subspace is derived. This will help analysts infer the potential impact of a change in the query parameters that defines the queried subspace. The parameters defining a subspace are essentially the predicates in an AQ. A convenient way to represent how a statistic is derived (in terms of compactly and succinctly conveying rich information) is by adopting a statistical regression function. A regression function describes how an output depends upon independent variables (input) and shows the contribution of each one of those variables to the output. Let y be the output and $\mathbf{x} \in \mathbb{R}^d$ be the independent variables. Then, using linear regression we can obtain the function $y = \mathbf{w}^T\mathbf{x} + b$, where $\mathbf{w} \in \mathbb{R}^d$ are the coefficients which dictate how the output y changes with respect to the input $\mathbf{x}$ and $b \in \mathbb{R}$ is the intercept. It should be noted here that in the model definition of the regression function, we add the inherent Gaussian noise which is independent on the independent variables; which we deliberately omit for reasons of simplicity in light of introducing the concept of Query-Driven explanations.

For example, we can derive a regression function that explains how the average (mean) household income for each region is generated, based on the size of the region. Thus, the resulting function can inform the data scientists as to how influential the size of a region is and how different results are generated across different subregions. This functionality is expected to allow the discovery of interesting patterns during exploratory analytics. However, deriving such regression functions is non-trivial. In this chapter, we propose a novel framework based on QDL, which explains *how* AQ results are derived.

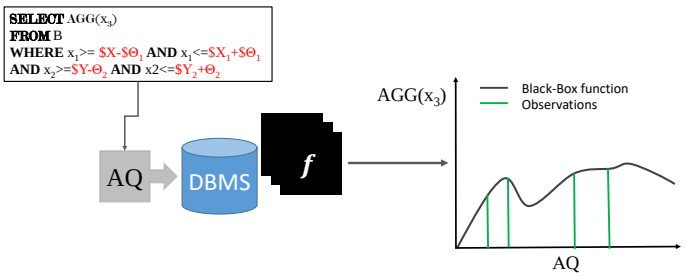

Figure 3.1: The range predicates in an AQ represent the input parameters of a black-box function that generates its result.

3.2 Problem Defintion & Motivating Examples

Consider that every result of an AQ is obtained by a black-box aggregate function (also called the *true* function) as shown in Figure 3.1. An initial SQL query (AQ) is issued and is processed by the DBMS. This AQ defines a region bounding dimensions/attributes (x_1, x_2). The contained rows residing in that region are used to compute an aggregate/statistic on dimension x_3. In this particular example we attempt to explain how the aggregation function over x_3 varies with the x_1 and x_2. Each aggregate function in an AQ can be represented by a different black-box function that accepts a set of input parameters and maps to a result. The structural form of this black-box function is *unknown*. We can only observe the result given a set of inputs as shown in Figure 3.1. In this case, the inputs are query parameters defining an AQ. Given a number of such observations (indicated by green vertical lines in Figure 3.1), we can adopt QDL to approximate the structural form of an aggregate function, using models that are highly interpretable (such as linear regression). However, the challenges are manyfold :

1. Identifying valid, accurate and interpretable models; The models need to have a valid structural form, they have to accurately mimic the *true* function in their responses and lastly they need to have an interpretable structure that an analyst can understand.

2. Placing models over the right subspaces to maximize accuracy; Over the next sections, we will describe how global models aiming to explain a complete data space fall short, and how we mitigated this problem by building local models over smaller regions.

3. As new queries are processed we need to continue updating the models to reflect changes in aggregate responses.

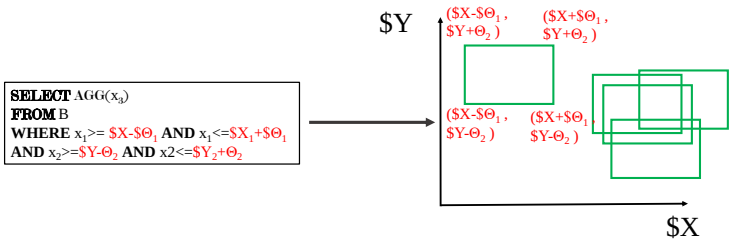

Figure 3.2: A number of AQs represented as rectangles in the 2D space. Each coordinate is given as a sum of the corresponding dimension and its length.

4. Identifying the right metrics to holistically evaluate the suggested approach.

The challenges mentioned above and additional obstacles that we encountered, while addressing these challenges, are discussed in the following sections. However, to make our case more concrete, we present some motivating examples that should help the reader throughout this chapter.

3.2.1 Motivating Examples

We focus on AQs with a *range* selection operator because of their wide-applicability in analytics tasks. A range operator is defined by a hyper-rectangle in multi-dimensional space. An example of a series of AQs is shown in Figure 3.2. The range predicates of an AQ are used to represent rectangles. Such operator is evident in many applications including: location-based search, e.g searching for spatially-close objects, such as astronomical objects, objects within a geographical region etc.

Example 1: Crimes Data. Consider analyzing a dataset, containing recorded incidents with attributes such as their location and the type of incident (homicide, burglary, etc.). One such example dataset is the Chicago Crimes Dataset [6]. A typical exploration query is to issue an AQ with a range operator:

```
SELECT COUNT(*) AS y FROM Crimes AS C
WHERE C.X > $X − $Θ1 AND C.X < $X + $Θ1 AND
C.Y > $Y − $Θ2 AND C.Y < $Y + $Θ2;
```

This query can be represented by a rectangle with a 2D center and two variables representing the length of the first and second dimension. The 2D center of the rectangle is defined by the

point ($X, $Y) and length by $\$\Theta_1$, $\$\Theta_2$ in both dimensions. Such AQ returns the number of incidents in a specific area of interest corresponding to an arbitrary neighborhood; see Figure 3.2. Using several such AQs, we can build regression functions with input variables the AQ parameters: ($X, $Y, $\$\Theta_1$, $\$\Theta_2$) and output y, i.e., the count statistic. The estimated regression coefficients would then provide a way for the analysts to infer the potential impact query parameters will have on the outcome. Therefore, the analysts have an understanding of which AQs will provide answers tailored to their interests. For instance, if they are interested in high crime index (increased output y), then the regression coefficients can inform them as to which query to execute next by changing the corresponding query parameters.

Example 2: Telecommunication Calls Data. Consider a data scientist tasked with identifying time-frames having high average call times. They need to issue AQs of varying-sized ranges over time, such as the following range-AQ:

```
SELECT AVG(Call_Time) AS y FROM TCD AS C
WHERE C.Time BETWEEN
$X − $Θ AND $X + $Θ
```

Discovering the aforementioned time-frames, without our proposed explanations, can be a daunting task as multiple queries have to be issued, overflowing the system with a number of AQs. The queries, could take minutes or hours to execute depending on the data size and throughput of the system. By using our proposed explanation function, the analyst could carry out their task with highly accurate answers. This is achieved by plugging in different query parameters to the given explanation function. Beyond that, analysts could formulate an optimization problem that could be solved using our methodology. Given a differentiable aggregation function, the maxima and minima points can also be estimated, thus, the analysts can easily discover the query parameters at which the AQ result is maximized or minimized. Again, such functionality is very much lacking and is crucial for exploratory analytics.

3.3 Background & Related Work

Our overarching aim is to provide explanations for AQs, whose efficient computation has been a major research interest [63, 33, 37, 142, 65, 20, 19, 137, 12, 99], with methods applying sampling, synopses and ML models to compute such queries. Compared to our work, the above works are largely complementary. One distinguishing feature is that our primary task is to *explain the AQ results* and do so efficiently, accurately, and scalably with respect to increasing data sizes.

Explanations for AQs can be linked with the well-known topic of tracking provenance in query results. Cheney et al. [36] describe the notion of *Data Provenance*. Specifically, they

mention three kinds of definitions under the umbrella of *Data Provenance*:

1. *Why Provenance* : Why a result is what it is.
2. *Where Provenance* : Where did the result come from ? With the answer often being the location of rows in a dataset, either on disk or the row number.
3. *How Provenance* : How was the result generated ?

To answer such questions, researchers usually track the underlying data contributing to the result. Thus, by creating a summary of the underlying data contributing to the result, they are able to construct an answer to the various provenance questions. A number of authors have extended this and introduced the concept of explanations. Their aim is to construct explanations on a variety of domains to help users gain insight for various tasks. To this end, the main objective of this chapter is to explore ways to assist analysts in understanding and analyzing subspaces by explaining AQs as efficiently and effectively as possible by utilizing QDL.

Explanation techniques have emerged in multiple contexts within the Data Management and ML communities. One of such contexts is to provide explanations, represented as predicates, for simple query answers as in [43, 108, 91]. Similarly, other authors have extended this to probabilistic and scientific databases [68], [141]. Explanations have also been used for interpreting outliers in both *in-situ* data [140] and in streaming data [27]. The authors first detect outliers, either manually or automatically, and then generate predicates or attribute-value combinations that explain the outliers set. In [94], the authors built a system to provide interpretations for service errors and present the explanations visually to assist debugging large scale systems. In addition, the authors of PerfXPlain [73] created a tool to assist users while debugging performance issues in Map-Reduce jobs. Other frameworks, provide explanations utilized by users, to locate any discrepancies found in their data [134], [31], [135]. A recent trend, is in explaining ML models for debugging purposes [78] or to understand how a model makes predictions [105] and conveys trust to users using these predictions.

Given the above, one can detect three central pillars emerging around the concept of explanations. When working with explanations, one has to determine the (i) domain, (ii) the scalability and efficiency of the approach in generating explanations, and (iii) how the explanation is represented. For instance, in [27] the domain is in *outliers analysis*, the explanations are represented using *attribute-value combinations*, and the approach is to use *statistical structures* (heavy hitter sketches [37]) that allow the analysis of streaming values.

For the work described in this chapter, we focused on explanations for AQs because of the AQs' wide use in exploratory analytics [137]. Hence the domain is *AQ explanation* within the context of exploratory data analysis. Both [140] and [16] focused on explaining aggregate

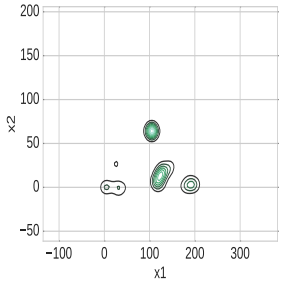

Figure 3.3: Real workload cluster analysis (Source SDSS [128]); x_1 and x_2 are parameters of a range query. The range queries form clusters meaning that intra-cluster queries are similar, thus can be explained by similar explanation functions.

queries. However, the former focused on explaining the existence of outliers in aggregate queries and the latter on tracking the '*how?*' provenance of AQs using semi-ring formalism. Our major difference is that AQs are explained solely based on the query input parameters and query results of previously issued and incoming queries. Thus, there is no direct data access, which both works do [140, 16]. Direct data access, makes generating explanations slow and inefficient for large-scale datasets. In addition, using the formalisms from [16] is impossible, within our defined context, as these formalisms, would be incomprehensible and in need of a provenance query language, as explicitly stated in [16].

Furthermore, scalability and efficiency are particularly important. Computing explanations is proved to be an NP-Hard problem [134] and proposed methods can take a long time [140, 108, 43] even with modest datasets. An exponential increase in data size implies a dramatic increase in the time to generate explanations. The framework proposed in this chapter does not suffer from these limitations and is able to construct explanations in milliseconds even with massive data volumes. This is achieved due to two principles: First, on workload characteristics. Workloads contain a large number of overlapping queried data subspaces, which has been acknowledged and exploited by recent research e.g., [137],[99], STRAT[32], and SciBORQ [123] and has been found to hold in real-world workloads involving exploratory/statistical analysis, such as in the Sloan Digital Sky Survey and in the workload of SQLShare [67]. This fact is shown in Figure 3.3. Using the Sloan Digital Sky Survey we extracted a number of range queries and their query parameters. In Figure 3.3, we have plotted the values found in the range queries and denote the existence of clusters around certain values. This fact leads us to believe the existence of overlapping ranges in queries and the existence of disjoint sets of queries that might share similar underlying black-box functions.

Based on this fact our second key principle is that, we rely on a novel framework that exploits the workload characteristics to optimally identify valid AQ explanation functions in an efficient and accurate manner.

The adopted explanation representation is in the form of regression functions. Recent works [92, 113] use this representation as it is highly parsimonious and can pinpoint the source of anomalies in aggregate results. In [92] this is used to explain data records and not AQs. In addition, past AQs are not considered for building regression functions. Therefore our work is applied within a different domain.

3.4 Explanation Representation

In this section we present the overall idea leading to the creation of the proposed framework. This acts as an introduction to the concept of explanations and brief overview of the suggested ML models that are used to produce explanations for aggregate queries. We start off by describing the queries and their answers, which we seek to explain. Then we move on to discuss about potential representations of the resulting explanations. Finally, we describe the process of finding an explanation for the query result and discuss different approaches.

3.4.1 Query Vectorial Representation

Let $\mathbf{a} = [a_1, \ldots, a_d] \in \mathbb{R}^d$ denote a random row vector (data point) in the d-dimensional data space $\mathbb{D} \subset \mathbb{R}^d$. A dataset $\mathcal{B}$ contains N random row data vectors $B = \{\mathbf{a}\}_{i=1}^{N}$; $|\mathcal{B}| = N$ indicates the cardinality of the set $\mathcal{B}$.

Definition 3.4.1. (Range Query) A range query is defined as the vector: $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$, $\mathbf{x} \in \mathbb{R}^d, \boldsymbol{\theta} \in \mathbb{R}^d_+$. This vector $\mathbf{q}$ defines a hyper-rectangle in multi-dimensional space as a series of conjunctions of predicates, i.e $\bigwedge_{i=1}^{d}(x_i - \theta_i \leq a_i \leq x_i + \theta_i)$.

Definition 3.4.2. (Data Subspace) Given a range query, a data subspace $\mathbb{D}(\mathbf{x}, \boldsymbol{\theta})$ is the convex subspace of $\mathbb{R}^d$, which includes data vectors $\mathbb{D}(\mathbf{x}, \boldsymbol{\theta}) = \{\mathbf{a} \in \mathbb{R}^d | \bigwedge_{i=1}^{d}(x_i - \theta_i \leq a_i \leq x_i + \theta_i)\}$.

Definition 3.4.3. (Query Similarity) The p-norm (L_p) distance between two query vectors $\mathbf{q}$ and $\mathbf{q}'$ from $\mathbb{R}^{2d}$ for $1 \leq p < \infty$, is $\|\mathbf{q} - \mathbf{q}'\|_p = (\sum_{i=1}^{d} |q_i - q'_i|^p)^{\frac{1}{p}}$ and for $p = \infty$, is $\|\mathbf{q} - \mathbf{q}'\|_\infty = \max_{i=1,\ldots,d}\{|q_i - q'_i|\}$.

Definition 3.4.4. (Aggregate Query) Given a data subspace $\mathbb{D}(\mathbf{x}, \boldsymbol{\theta})$ an AQ $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$ with input center $\mathbf{x}$ and vectorial length $\boldsymbol{\theta}$, an aggregate function, is represented via a regression function $f : \mathbb{R}^d \times \mathbb{R}^d_+ \to \mathbb{R}$ over $\mathbb{D}(\mathbf{x}, \boldsymbol{\theta})$, that produces a query response variable $y = f(\mathbf{x}, \boldsymbol{\theta})$ which is the result of an AQ. We notate with $\mathbb{Q} \subset \mathbb{R}^{2d}$ the query vectorial space.

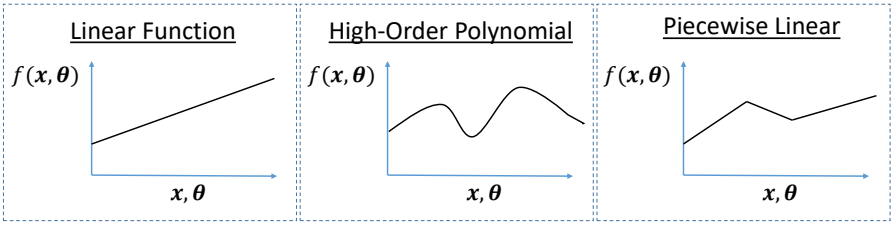

Figure 3.4: Different Types of Explanation Functions.

3.4.2 Functional Representation of Explanations

The defined AQ returns a single scalar value $y = f(\mathbf{x}, \boldsymbol{\theta})$ to the analysts. We seek to explain how such values are generated by finding a *function* $f : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ that can describe how y is produced given an ad-hoc query $\mathbf{q}$. We first discuss possible forms of the said function.

An approximate explanation function can be linear or high-order polynomial to approximate non-linear functions. When using high-order polynomial functions for our explanations, the analyst can no longer interpret the structural form of the model because of all the added terms. In addition, by using high-order polynomials we make the assumption that the order of the polynomial is known. On the other hand, adopting a single linear regression function will be an inaccurate representation of f. A linear regression function, increases/decreases infinitely as the input increases/decreases. Therefore, it is impossible to accurately represent non-linear functions with this choice.

Therefore, choosing the right family of functions is non-trivial. We choose to employ multiple locally-linear functions to capture the possible inherent non-linearity of f as we vary the input query parameters, i.e., $\mathbf{x}$ and $\boldsymbol{\theta}$ vectors.

We use a particular family of locally-linear functions, called Piecewise-Linear Regression (PLR). The approximation of f, using PLR, addresses the above shortcomings by finding the *best*, locally-linear regression functions. Using PLR, we can approximate *true* functions which are linear or non-linear in a coarse grained manner that is more interpretable than a high-order polynomial. The analyst is simply exposed to the different locally-linear function under a given segment. Examples of all three families of functions considered, are shown in Figure 3.4. As witnessed from Figure 3.4, PLR is essentially composed of multiple linear functions.

Definition 3.4.5. (Explanation Function) Given an AQ $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$, an explanation function $f(\mathbf{x}, \boldsymbol{\theta})$ is defined as the fusion of local piecewise linear regression functions $f \approx \sum \hat{f}(\mathbf{x}, \boldsymbol{\theta})$ derived by the fitting of the local functions $\hat{f}$ over similar previously executed AQ queries $\mathbf{q}' = (\mathbf{x}', \boldsymbol{\theta}')$ to the AQ $\mathbf{q}$.

Although we make the choice of using PLR functions because of their high interpretability and flexibility to fit both linear and non-linear functions, our framework can also use other ML models that can provide a method of estimating the importance of the input parameters. The optimization problems, to be defined in the upcoming sections, can be adapted to work with such models. However, as we will elaborate later, the importance of the query parameters only show the relative importance at a particular subspace and not necessarily how the output will change with respect to a change in the input query parameters.

3.5 Explanation Approximation Fundamentals

The challenge in approximating the underlying function f over the data subspaces defined by an AQ, lies in seeking local regression functions. These functions should explain the way query result y varies as query vector $\mathbf{q}$ changes without access to the underlying data, as this would harm efficiency. We build explanation functions by *only* leveraging previously executed and incoming AQs. In this section, we describe the methodology followed, for accurately identifying such PLR functions.

3.5.1 Explanation Approximation

We utilize AQs to train statistical learning models that are able to accurately approximate the *true* explanation function f for any possible query.

Formally, given a well defined explanation loss $\mathcal{L}(f, \hat{f})$ between the *true function* f and its *approximation* $\hat{f}$, we seek the optimal approximation function $\hat{f}^*$ that minimizes the Expected Explanation Loss (EEL) for *all possible* queries:

$$\hat{f}^* = \arg\min_{\hat{f} \in \mathcal{F}} \int_{\mathbf{x} \in \mathbb{R}^d} \int_{\boldsymbol{\theta} \in \mathbb{R}^d_+} \mathcal{L}(f(\mathbf{x}, \boldsymbol{\theta}), \hat{f}(\mathbf{x}, \boldsymbol{\theta})) p(\boldsymbol{\theta}, \mathbf{x}) d\boldsymbol{\theta} d\mathbf{x}, \tag{3.1}$$

where $\boldsymbol{\theta}$ is strictly positive as it defines the data subspace covered by the query's hyper-rectangle and $p(\mathbf{x}, \boldsymbol{\theta})$ is the probability density function of the query vectors $\mathbf{q} \in \mathbb{Q}$. Eq(3.1) is an optimization problem, where its solution gives us the optimal approximation of the true underlying function.

However, as stated earlier, accuracy will be problematic as it seems intuitively wrong that a single function, $\hat{f}^*$, can explain *all* queries at an arbitrary location $\mathbf{x}$ with an arbitrary length $\boldsymbol{\theta}$. Such function is innaccurate because analysts issue queries over different subspaces of interest. Consider the *Crimes Data* example, data analysts might issue AQs with a different length and a fixed center, to compare whether a statistic over a small neighbourhood increases as the neighbourhood size varies. The result of these AQs, might vary abruptly. In addition, having a fixed length and a varying center $\mathbf{x}$ will almost surely produce different output as the analysts are essentially querying different fixed size areas. Even with the use of PLRs, we found that variance in the output was still large enough that the produced explanations were not accurate enough. Therefore, having a single function acting as a global explanation for all possible queries is not an ideal approach.

Hence, we introduce *local* approximation functions $\hat{f}_1, \ldots, \hat{f}_K$ that collectively minimize the objective in (3.1). Thus, the objective is no longer to find a global approximation to the *true* function for all possible queries. Instead, we fit a number of locally optimal functions, where each of them can explain a *subset* of possible queries. We refer to those models as Local PLR Model (LPM)s.

LPMs, are fitted using AQs forming a cluster of similar query parameters. For each uncovered cluster of AQs, we fit (at least) an LPM. Therefore, if K clusters are obtained from clustering the query vectorial space $\mathbb{Q}$, then K LPMs are fitted. Intuitively, this approach utilizes queries that are *similar* to each other within the same cluster. It relies on the hypothesis that these queries will tend to have *similar* results. So in turn, the underlying *true* function generating their outputs will have *similar* statistical structure. It will be shown that the resulting fused explanation is more accurate, as the reduced variance in the input query parameters effectively reduces the variance in the result y. This was empirically shown from our experimental workload.

Formally, to minimize the EEL, we seek K local approximation functions $\hat{f}_k \in \mathcal{F}, k \in [K]$ from a family of linear regression functions $\mathcal{F}$, such that for each query $\mathbf{q}$ belonging to the partition/cluster k of the query space, notated as $\mathbb{Q}_k$, the summation of the local EEL is minimized:

$$\mathcal{J}_0(\{\hat{f}_k\}) = \sum_{\hat{f}_k \in \mathcal{F}} \int_{\mathbf{q} \in \mathbb{Q}_k \subset \mathbb{R}^{2d}} \mathcal{L}(f(\mathbf{q}), \hat{f}_k(\mathbf{q})) p_k(\mathbf{q}) d\mathbf{q} \tag{3.2}$$

where $p_k(\mathbf{q})$ is the probability density function of the query vectors belonging to a query subspace $\mathbb{Q}_k$. Thus, $\mathcal{J}_0$ forms a *generic* optimization problem. **Note**: The *explanation loss* $\mathcal{L}(f, \hat{f})$ represents the discrepancy of the actual explanation function f due to the approximation of explanation $\hat{f}$. For evaluating the loss $\mathcal{L}$, we propose two different aspects: (1) *the statistical aspect*, where the goodness of fit of the explanation function is measured, and (2)

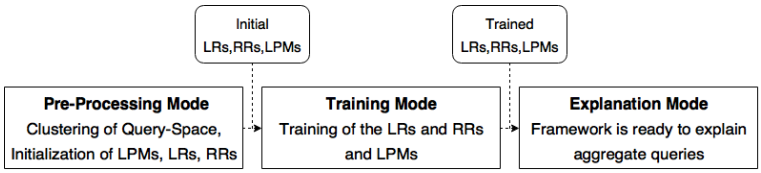

Figure 3.5: The three different modes of the proposed framework. Each resulting output from each mode is pipelined into the next.

the *predictive accuracy* denoting how well the results from the *true* explanation function can be approximated using the explanation function; refer to Section 3.9 for these metrics.

3.5.2 Framework Overview

The proposed methodology for computing explanations is split into three *modes* (Figure 3.5). The *Pre-Processing Mode* identifies the *optimal* number of LPMs and an initial approximation of their parameters using previously executed queries. The additonal, elements suggested by Figure 3.5 are explained in the following sections. The purpose of *Pre-Processing Mode* is to jump-start our framework. In *Training Mode*, the LPMs' parameters are incrementally optimized to minimize the objective function (3.2) as incoming queries are processed in an on-line manner. In *Explanation Mode*, the framework is ready to explain AQ results via the obtained LPMs.

Pre-Processing Mode

A training set $\mathcal{T} = \{(\mathbf{q}, y)_1, \ldots, (\mathbf{q}, y)_m\}$ of $|\mathcal{T}| = m$ previously executed queries $\mathbf{q}$ and their corresponding results, y, is used as input to the *Pre-Processing Mode*. The central task, is to cluster/partition the query space $\mathbb{Q}$ based on the observed previous queries $\mathbf{q} \in \mathcal{T}$ into K clusters, also referred to as subspaces $\mathbb{Q}_k$. Within each cluster, queries with similar centers $\mathbf{x}$ are grouped together. Each cluster is then further partitioned into L sub-clusters, as queries with similar centers $\mathbf{x}$ are separated by their $\boldsymbol{\theta}$ parameter values. Therefore, the approach followed, is a hierarchical query space partitioning, with the first level partition, with respect to center $\mathbf{x}$ and second level partition with respect to parameter $\boldsymbol{\theta}$. Where each Level-1 (L1) cluster $\mathbb{Q}_k, k = 1, \ldots, K$ is associated with a number of Level-2 (L2) sub-clusters $\mathbb{U}_{kl}, l = 1, \ldots, L$.

For each L1 cluster $\mathbb{Q}_k$ and L2 sub-cluster $\mathbb{U}_{kl}$, we assign an L1 representative, hereinafter referred to as Location Representative (LR) and an L2 representative, hereinafter referred to as Region Representative (RR). The LR converges to the mean vector of the centers of all queries belonging to L1 cluster $\mathbb{Q}_k$, while the associated RR converges to the mean length

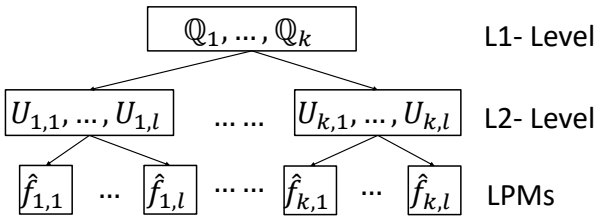

Figure 3.6: The hierarchical quantization scheme provides levels of partitioning for the multi-dimensional center $\mathbf{x}$ and region lengths $\boldsymbol{\theta}$. Each LPM is associated and trained with the data points included in a cluster in L2.

value of the lengths of all queries, whose lengths values belong to $\mathbb{U}_{kl}$. After the hierarchical partitioning of the query space, the task is to *associate an LPM* $\hat{f}_{kl}(\mathbf{q})$ with *each* L2 sub-cluster $\mathbb{U}_{kl}$. This process is nicely summarized in Figure 3.6. At L1 we have the partitioned Query Space $\mathbb{Q} = \mathbb{Q}_1 \cup \mathbb{Q}_2, \ldots, \mathbb{Q}_{k-1} \cup \mathbb{Q}_k$. Each one of the partitions is associated with a sub-cluster at L2 which are also associated with an LPM.

Training Mode

This mode optimally adapts the parameters of LRs and RRs, obtained from the *Pre-Processing mode*, in order to minimize the objective function in (3.2). This optimization process is achieved incrementally by processing each new pair $(\mathbf{q}_i, y_i)$ in an on-line manner. Consulting Figure 3.7, in *Training* mode, each incoming query $\mathbf{q}_i$ is mapped to the closest LR corresponding to an L1 cluster. Since, the closest LR is associated with a number of RRs, the query is assigned to one of those RRs, and the associated representatives are adapted. After a pre-specified number of processed queries, the corresponding LPM $\hat{f}_{kl}(\mathbf{q})$ is re-adjusted to account for newly associated queries.

Explanation Mode

In this mode, no more modifications to LRs, RRs, and the associated LPMs are made. Based on the L1/L2 representatives and their associated LPMs, the model can explain AQs. Figure 3.7 sums up the result of all three modes and how an explanation is given to the user. For a given query $\mathbf{q}$, the closest LR;$\mathbf{w}_k$ is initially obtained and then, based on a combination of the RRs;$(\mathbf{u}_{k,1}, \ldots, \mathbf{u}_{k,3})$ and their associated LPMs;$(\hat{f}_{k,1} \ldots, \hat{f}_{k,3})$, returns an explanation as

a fusion of diverse LPMs functions derived by the L2 level. We elaborate on this fusion of L2 LPMs in Section 3.8.

3.6 Optimization Problems Deconstruction

In this section, we deconstruct the generic optimization problem described in Eq. 3.2 into two distinct phases which can be optimized individually. We then explain how all phases fit back together. In addition, all three introduced modes, *Pre-Processing*, *Training*, *Explanation*, are discussed in more detail and we elaborate on the specific algorithms used for each task.

3.6.1 Optimization Problem 1: Query Space Clustering

The first part of the deconstructed generic problem solves the need to find *optimal* LRs and RRs, as such optimal parameters guarantee better grouping of queries thus better approximation of *true* function, during the *Pre-Processing* mode. The LRs are initially random location vectors $\mathbf{w}_k \in \mathbb{R}^d, k = 1, \ldots, K$, and are iteratively refined by a clustering algorithm until they converge to the mean vector of the associated query space $\mathbb{Q}_k$. Formally, this method converges to the optimal mean vectors $\mathcal{W} = \{\mathbf{w}_k\}_{k=1}^{K}$, which minimize the L1 Expected Quantization Error (L1-EQE):

$$\mathcal{J}_1(\{\mathbf{w}_k\}) = \mathbb{E}[\|\mathbf{x} - \mathbf{w}^*\|^2; \mathbf{w}^* = \arg\min_{k=1,\ldots,K} \|\mathbf{x} - \mathbf{w}_k\|^2] \tag{3.3}$$

where $\mathbf{x}$ is the location of query $\mathbf{q} \in \mathcal{T}$ and $\mathbf{w}_k$ is the mean center vector of all queries $\mathbf{q} \in \mathbb{Q}_k$ associated with $\mathbf{w}_k$. We adopt the K-Means [61] clustering algorithm to identify the LRs based on the queries' centers $\mathbf{x}$. The choice of K-Means is mainly due to its scalability to many training examples and its simplicity. A limitation of the K-Means algorithm is the need to specify parameter K, which is the number of LR representatives. Therefore, we devised a simple strategy to find a near-optimal K. By running the clustering algorithm iteratively, each time increasing the input parameter K for the K-Means algorithm, we are able to find a K that is near-optimal. In this case, an optimal K would *sufficiently* minimize the Sum of Squared Quantization Errors (SSQE), which is equal to the summation of distances, of all queries from their respective LRs. The strategy is elaborated in Algorithm 1.

$$\text{SSQE} = \sum_{i}^{n} \min_{\mathbf{w}_k \in \mathcal{W}} (||\mathbf{x}_i - \mathbf{w}_k||_2^2) \tag{3.4}$$

The algorithm is fairly straight-forward and is in-line with the *"Elbow Method"* in approximating an optimal K for K-means. It essentially performs several passes over the data,

applying the algorithm and obtaining an SSQE. It stops when a pre-defined threshold $\epsilon > 0$ has been reached. We note that there are multiple such algorithms available in the literature [60]. However, this is not part of our focused work thus a simple solution was preferred to alleviate this problem.

Algorithm 1 Estimating a near-optimal K

```
Input: ϵ; K                                   ▷ initial K; predefined improvement threshold.
W = ∅; X = {x_i}_{i=1}^m : x ∈ T              ▷ set of LRs; query centers T
while TRUE do
    W ← KMeans(K, X)                          ▷ call K-Means algorithm with K LRs
    SSQE ← Σ_{i=1}^m min_{w_k ∈ W} (||x_i − w_k||^2)   ▷ Calculate SSQE
    if Δ|SSQE| > ϵ then                       ▷ improvement
        K ← K + 1                             ▷ increase K
    else
        break                                 ▷ no more improvement; exit
    end if
end while
Return: W                                     ▷ set of K LRs
```

We also utilize K-Means over each L2 cluster of queries created by the L1 query partitioning phase. Formally, the objective is to minimize the conditional L2 Expected Quantization Error (L2-EQE):

$$\mathcal{J}_{1.1}(\{\mathbf{u}_{k,l}\}) = \mathbb{E}[\|\boldsymbol{\theta} - \boldsymbol{u}^*\|^2; \mathbf{u}^* = \arg\min_{l=1,\ldots,L} \|\boldsymbol{\theta} - \mathbf{u}_l\|^2], \tag{3.5}$$

Therefore, for each LR $\mathbf{w}_1, \ldots, \mathbf{w}_K$, we *locally* run the K-Means algorithm with L number of RRs, where a near optimal value for L is obtained following the same near-optimal strategy outlined in Algorithm 1. With SSQE being computed using $\mathbf{u}$ and $\boldsymbol{\theta}$ instead of $(\mathbf{w}, \mathbf{x})$. Specifically, we identify the L2 RRs over the lengths of those queries from $\mathcal{T}$ whose closest LR is $\mathbf{w}_i$. Then, by executing the L-Means over the length values from those queries we derive the corresponding set of region representatives $\mathcal{U}_i = \{\mathbf{u}_{i1}, \ldots, \mathbf{u}_{iL}\}$, where each $\mathbf{u}_{il}$ is the mean vector of lengths in the l-th L2 sub-cluster of the i-th L1 cluster. Thus the first part of the deconstructed optimization problem can be considered as two-fold, as we wish to find optimal parameters for both LRs and RRs that minimize (3.3) and (3.5).

3.6.2 Optimization Problem 2: Fitting LPMs per Query Cluster

The second part of the deconstructed generic optimization problem has to do with fitting *optimal* functions such that the local EEL is minimized given the optimal parameters obtained from the first part of the deconstructed problem. We fit PLR functions $\hat{f}_{kl}(\mathbf{q})$ for each L2 sub-cluster $\mathbb{U}_{kl}$. The fitted PLR, captures the *local* statistical dependency of the input param-

eters, given that $\mathbf{x}$ is a member of the L1 cluster represented by $\mathbf{w}_k$ and $\boldsymbol{\theta}$ of L2 represented by $\mathbf{u}_{kl}$. Given the objective in (3.2), for each local L2 sub-cluster, the approximate function $\hat{f}_{kl}$ minimizes the conditional Local EEL:

$$\begin{aligned} \mathcal{J}_2(\{\beta_{kl}, \lambda_{kl}\}) &= \mathbb{E}_{\theta,\mathbf{x}}[\mathcal{L}(f_{kl}(\mathbf{q}), \hat{f}_{kl}(\mathbf{q}))] \\ \text{s.t.} \quad & \mathbf{w}_k = \arg\min_{j\in[K]} \|\mathbf{x} - \mathbf{w}_j\|_2^2, \\ & \mathbf{u}_{kl} = \arg\min_{j\in[L]} ||\boldsymbol{\theta} - \mathbf{u}_{kl}||_2^2 \end{aligned} \tag{3.6}$$

conditioned on the closeness of the query's $\mathbf{x}$ and $\boldsymbol{\theta}$ to the L1 and L2 partioned query space $\mathbb{Q}_k$ and $\mathbb{U}_{kl}$, respectively. Where $\{\beta_{kl}, \lambda_{kl}\}$ are the parameters of the PLR function $\hat{f}_{kl}$ defined in (3.7).

Remark 1: Minimizing objective $\mathcal{J}_2$ in (3.6) is not trivial due to the double conditional expectation over each query center and length. To *initially* minimize this local objective, we adopt Multivariate Adaptive Regression Splines (MARS) [48] as the approximate model explanation function $\hat{f}_{kl}$. Thus, $\hat{f}_{kl}$ has the following form:

$$\hat{f}_{kl}(\mathbf{q}) = \beta_0 + \sum_{i=1}^{M} \beta_i h_i(\mathbf{q}), \tag{3.7}$$

where $h_i(\mathbf{q})$ are basis functions identified by a forward stepwise procedure. Essentially, this creates M regression functions. The number M of linear regression functions is automatically derived by MARS using a threshold for convergence with respect to R^2(*coefficient-of-determination*, later defined); which optimize fitting. Thus, guaranteeing an optimal number of M linear regression functions. In total, $K \times L$ MARS functions, are used, for providing explanations for the whole query space. Figure 3.7 illustrates the two levels L1 and L2 of our explanation methodology, where each LR and RR are associated with a MARS model.

Using alternatives to LPMs

As described, we use MARS, which empirically performs really well and also has desirable properties. We can derive the importance of features by their use in basis functions and their coefficients. Because of its building procedure it can also eliminate terms that do not increase predictive power. Thus the analyst can infer which parameters are not crucial in producing the output of aggregate functions. However, the model choice should not be restrictive. Alternative models that provide similar functionality can be used as well. Regression trees [83] can also provide similar functionality as the importance of each parameter can be inferred by its use in branches. In addition, simple Linear Regression models and their variants Ridge [62], Lasso [47] are also good candidates as their coefficients provide intuitive explanations.

Although in this case increased partitioning might be needed as the obtained clusters might be non-linear with respect to the input parameters and output, as described earlier.

3.6.3 Optimization Problem 3: Putting it All Together

The optimization objective functions in (3.3), (3.5) and (3.6) are combined to establish a generic optimization objective $\mathcal{J}_0$, which includes the estimation of *optimal* parameters that minimize the L1-EQE and L2-EQE in $\mathcal{J}_1$ (L1) and $\mathcal{J}_{1.1}$ (L2), and the conditional optimization of parameters in $\mathcal{J}_2$. In this context, we need to estimate the values of parameters in $\mathcal{W} = \{\mathbf{w}_k\}$ and $\mathcal{U} = \{u_{kl}\}$ that minimize the EEL given that our explanation comprises a set of regression functions $\hat{f}_{kl}$. The generic optimization objective is to identify *all* parameters from $\mathcal{J}_1$, $\mathcal{J}_{1.1}$, and $\mathcal{J}_2$ from Problems 1 and 2:

$$\mathcal{J}_3(\mathcal{W},\mathcal{U},\mathcal{M}) = \mathcal{J}_1(\mathcal{W}) + \mathcal{J}_{1.1}(\mathcal{U}) + \mathcal{J}_2(\mathcal{M}) \tag{3.8}$$

with parameters:

$$\mathcal{W} = \{\mathbf{w}_k\}, \mathcal{U} = \{\mathbf{u}_{kl}\}, \mathcal{M} = \{(\beta_i, \lambda_i)_{kl}\} \tag{3.9}$$

with $k \in [K]$, $l \in [L]$, $i \in [M]$, which can be adapted online (as explained later) and will be used for explaining AQ.

Remark 2: The optimization function $\mathcal{J}_3$ approximates the generic objective function $\mathcal{J}_0$ in (3.2) via L1 and L2 query partitioning (referring to the integral part of (3.2)) and via the estimation of the local PLR functions referring to the family of function space $\mathcal{F}$. Hence, we hereinafter contribute to an algorithmic solution to the optimization function $\mathcal{J}_3$ approximating the *theoretical* objective function $\mathcal{J}_0$.

3.7 Statistical Learning Methodology

In this section, a new statistical learning model is proposed, that associates the (hierarchically) partitioned query space with PLR-based explanation functions. Given the hierarchical query space partitioning and LPM fitting, the *Training Mode* fine-tunes parameters in (3.9) to optimize both $\mathcal{J}_1$, $\mathcal{J}_{1.1}$ in (3.3), (3.5) and $\mathcal{J}_2$ in (3.6). The three main sets of parameters $\mathcal{W}, \mathcal{U}$, and $\mathcal{M}$ of the framework are *incrementally* fine-tuned in *parallel* using queries issued to the underlying DBMS. These queries are issued online and the incremental process is as

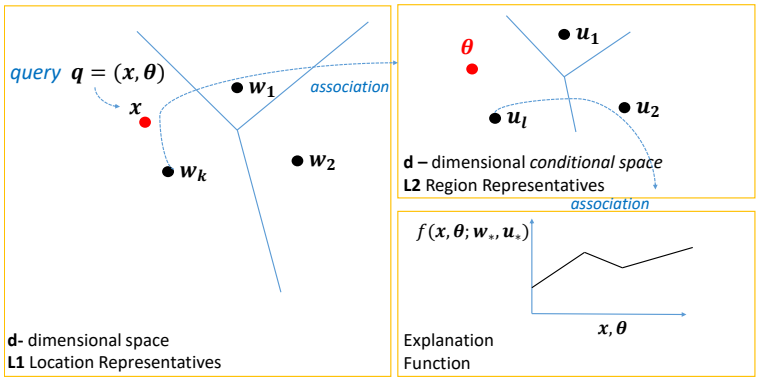

Figure 3.7: Rationale: Query mapping to L1 cluster, then mapping to L2 sub-cluster conditioned on the L1 LR, and association to an explanation function. The explanation is provided by the associated LPM.

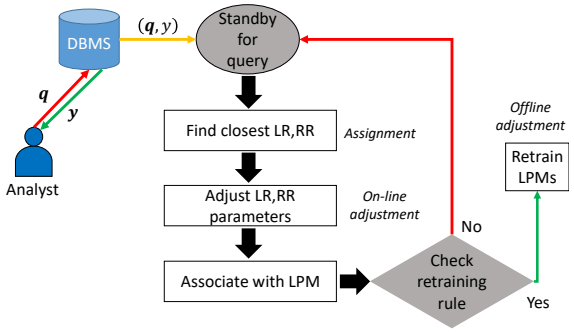

Figure 3.8: The *Training Mode* : A query-result pair is initially associated with an LR, RR and LPM. The LR, RR parameters are adjusted online as each query is processed. An offline adjustment step is triggered based on a threshold.

follows : (1) the analyst issues an AQ $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$; and (2) the DBMS answers with result y ; (3) our framework exploits pairs $(\mathbf{q}, y)$ to train its new statistical learning model.

The *Training Mode* uses three steps and the complete process is shown in Figure 3.8. Firstly, a query $\mathbf{q}$ is issued by an analyst to the DBMS and the query is associated with an L1, L2 and LPM, using the *assignment step* highlighted in Figure 3.8. Next an *On-line adjustment* step is performed, in which the LR and RR are gradually modified with respect to, the associated incoming query in the direction of minimizing the said objectives. Finally, a *retraining rule* is checked and if the rule is satisfied an *off-line adjustment* step conditionally fine-tunes any LPM associated with any processed query during *Training mode*.

1. Query Assignment Step. For each executed query-answer pair $(\mathbf{q}, y)$, $\mathbf{q}$ is asscoiated with its closest LR using only the query center $\mathbf{x}$ based on (3.3). Concretely we use (3.10) to

identify the closest LR.

$$\mathbf{w}^* = \arg\min_{i \in K} ||\mathbf{w}_i - \mathbf{x}||_2^2 \tag{3.10}$$

Obtaining the closest LR, $\mathbf{w}_k^*$, allows us to *directly* retrieve the associated RRs $\mathcal{U}_k^* = \{\mathbf{u}_{1k}^*, \ldots, \mathbf{u}_{Lk}^*\}$. Finding the best RR in $\mathcal{U}_k^*$ is, however, more complex than locating the best LR. In choosing one of the RRs, we consider both *distance* and the associated *prediction error*. Specifically, the *prediction error* is obtained by the LPM of each RR from the set $\mathcal{U}_k^*$. Hence, in this context, we first need to consider the distance of query $\mathbf{q}$ to all of the RRs in $\mathcal{U}_k^*$:

$$||\boldsymbol{\theta} - \mathbf{u}_{kl}||_1, \forall \mathbf{u}_{kl} \in \mathcal{U}_k^*, \tag{3.11}$$

and, also, the *prediction error* given by each RR's associated LPM $\hat{f}_{kl}$. The prediction error is obtained by the squared difference of the actual result y and the predicted outcome of the LPM, $\hat{y} = \hat{f}_{kl}(\mathbf{q})$:

$$(y - \hat{f}_{kl}(\mathbf{q}))^2, l = 1, \ldots, L \tag{3.12}$$

Therefore, to assign a query $\mathbf{q}$ to an RR, we combine both distances in (3.11) and (3.12) to get the assignment distance in (3.13), which returns the RR in $\mathcal{U}_k^*$ which minimizes:

$$l^* = \arg\min_{l \in [L]} \{z||\boldsymbol{\theta} - \mathbf{u}_{kl}||_1 + (1 - z)(y - \hat{f}_{kl}(\mathbf{q}))^2\} \tag{3.13}$$

The parameter $z \in (0, 1)$ tilts our decision towards the *distance*-wise metric or the *prediction*-wise metric, depending on which aspect we wish to attach greater significance.

Remark 3: *Why incorporate prediction error?* We could associate an incoming query with the closest RR as is done with an LR. However, note that an explanation function may have lower prediction error even though is not the closest (with respect to RR). Intuitively, this holds true, as some function might be able to make better generalizations even if their RRs are farther apart. Therefore, we introduce the weighted-distance in (3.13) to account for this and make more sophisticated selections.

2. On-line Representatives Adjustment Step. This step optimally adjusts the positions of the chosen LR and RR so that these parameters are informed by new queries. Their positions are shifted using SGD [30] over $\mathcal{J}_1$ and $\mathcal{J}_2$ with respect to $\mathbf{w}$ and $\boldsymbol{\theta}$ parameters in the negative direction of their gradients, respectively. This ensures the optimization of both objective functions. Theorems 1 and 2 present the update rule for the RR selected in (3.13) to minimize the EEL given that a query is projected to its LR and its convergence to the median value of the length of those queries.

Theorem 1. Given a query $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$ projected onto the closest LR $\mathbf{w}_{k^*}$ and RR $\mathbf{u}_{k^*,l^*}$, the

update rule for $\mathbf{u}_{k^*,l^*}$ that minimizes $\mathcal{J}_2$ is:

$$\Delta\mathbf{u}_{k^*,l^*} \leftarrow \alpha z \text{sgn}(\boldsymbol{\theta} - \mathbf{u}_{k^*,l^*}) \tag{3.14}$$

Proof. We adopt the Robbins-Monro stochastic approximation for minimizing the combined distance-wise and prediction-wise loss given an RR u, that is minimizing $\mathcal{E}(\mathbf{u}) = z\|\boldsymbol{\theta} - \mathbf{u}\|_1 + (1-z)(y - \hat{f}(\boldsymbol{\theta}; \mathbf{w}))^2$, using SGD over $\mathcal{E}(u)$. Given the t-th training pair $(\mathbf{q}(t), y(t))$, the stochastic sample $E(t)$ of $\mathcal{E}(\mathbf{u})$ has to decrease at each new pair at t by descending in the direction of its negative gradient with respect to $\mathbf{u}(t)$. Hence, the update rule for RR u is derived by:

$$\Delta\mathbf{u}(t) = -\alpha(t)\frac{\partial E(t)}{\partial \mathbf{u}(t)},$$

where scalar $\alpha(t)$ satisfies $\sum_{t=0}^{\infty}\alpha(t) = \infty$ and $\sum_{t=0}^{\infty}\alpha(t) < \infty$. From the partial derivative of $E(t)$ we obtain $\Delta\mathbf{u} \leftarrow \alpha z \text{sgn}(\boldsymbol{\theta} - \mathbf{u})$. By starting with arbitrary initial training pair $(\mathbf{q}(0), y(0))$, the sequence $\{\mathbf{u}(t)\}$ converges to optimal RR $\mathbf{u}$ parameters. □

$\alpha \in (0, 1)$ is the learning rate defining the shift of $\boldsymbol{\theta}$ ensuring convergence to optimal position and $sgn(x) = \frac{d|x|}{dx}, x \neq 0$ is the signum function. Given that query $\mathbf{q}$ is projected on an LR $\mathbf{w}_k^*$ and on an RR, $\mathbf{u}_{k^*,l^*}$, the corresponding RR converges to the local median of all length values of those queries.

Theorem 2. Given the optimal update rule in (3.14) for an RR $\mathbf{u}_{k,l}$, it converges to the median of the $\boldsymbol{\theta}$ values of each dimension of those queries projected onto the L1 query subspace $\mathbb{Q}_k$ and the L2 sub-cluster $\mathbb{U}_{kl}$, i.e., for each query $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$ with $\mathbf{x} \in \mathbb{Q}_k$, it holds true for $\mathbf{u}_{kl} \in \mathbb{U}_{kl}$ that: $\int_0^{u_{kl}} p(\theta|\mathbf{w}_k)d\theta = \frac{1}{2}$.

Proof. Focus on one dimesion, and consider the optimal update rule in (3.14) for an RR u and suppose that u has reached equilibrium, i.e., $\Delta u = 0$ holds with probability 1. By taking the expectations of both sides and replacing Δu with the update rule from Theorem 1:

$$\mathbb{E}[\Delta u] = \int_{\mathbb{R}} \text{sgn}(\theta - u)p(\theta)\text{d}\theta = P(\theta \geq u)\int_{\mathbb{R}} p(\theta)\text{d}\theta - P(\theta < u)\int_{\mathbb{R}} p(\theta)\text{d}\theta = 2P(\theta \geq u) - 1.$$

Since $\Delta u = 0$ thus u is constant, then $P(\theta \geq u) = \frac{1}{2}$, which denotes that u converges to the median of length for those queries represented by L1 RL and the associated L2 RR. □

Using SGD, $\boldsymbol{\theta}_{k^*,l^*}$ converges to the median of all length values of all queries in the local L2 sub-cluster in an on-line manner.

3. Off-line Adjustment Step. The mini-batch adjustment step is used to conditionally re-train the LPMs to reflect the changes by (3.14) in $\mathbb{U}_k$ parameters. As witnessed earlier, representatives are incrementally adjusted based on the projection of the incoming query-answer pair onto L1 and L2 levels. For the LPMs, the adjustment of hinge points and parameters (β_i, λ_i) needs to happen in mini-batch mode taking into consideration the projected incoming queries onto the L2 level. To achieve this, we keep track of the number of projected queries on each L2 sub-cluster $\mathbb{U}_k$ and re-train the corresponding LPMs $\hat{f}_{kl}$ given a conditionally optimal RR $\mathbf{u}_{kl}$. For every processed query we increment a counter. Once we reach a predefined number of projected queries-answers, we re-train every LPM that was affected by projected training pairs.

Remark 3: *Why Pre-Processing and Training Modes ?* A concrete explanation as to why these two modes need to co-exist was not provided in the previous sections. The curious reader might notice that with just the *Pre-Processing Mode* we could do a pretty good job at creating good enough LPMs to explain any new possible queries. However, with just a *Pre-Processing mode* we eliminate the possibility of adjusting as new query-answers pairs are processed. In addition, it does not take into consideration the prediction error thus it might lead to inaccurate models which only consider the similarity of query patterns and not the predictive power of LPMs. Moreover, *Training Mode* cannot exist on its own as the LPMs need a number of queries to be initialized hence the need for a *Pre-Processing* step.

3.8 Explanation Serving

After *Pre-Processing* and *Training*, explanations can be provided for *unseen* AQs, i.e., AQs which have never been used before during training or executed before over the DBMS data. The explanations are based on the associated LPM found using the AQ's input parameter values. The process of returning an explanation function to the user is as follows: firstly, the analyst issues a query $\mathbf{q}$, $\mathbf{q} \notin \mathcal{T}$, then, the closest LR, RR representatives are located, and finally, the associated LPM is returned to the analyst used for explanation, as will be discussed later.

As with *Training Mode*, the closest LR is identified using (3.10) and for the closest RR (3.13) is used. As mentioned, the L2 representatives are associated with an LPM and, thus, the related LPM is returned to the analyst.

This explanation function serves as an approximation of how the result of an unseen AQ varies in that subspace queried by the original AQ. The explanation function coefficients (or feature importance methods of various ML models) gives us a way to infer how the different query parameters contribute to the end AQ result. The suggested method in this chapter is able to do that by using only knowledge provided by coarse grained queries executed over

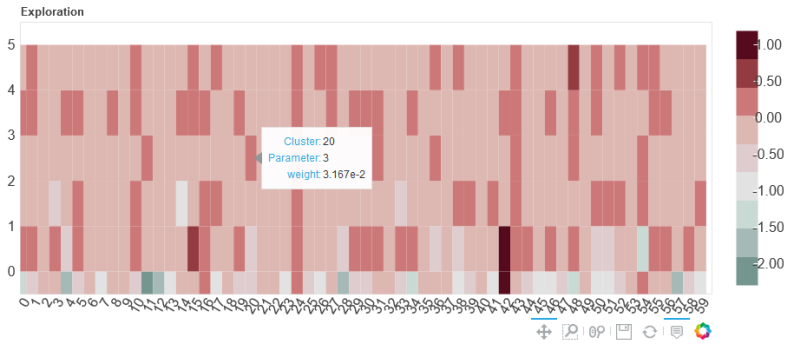

Figure 3.9: Interactive exploration of data spaces by visualising the importance of parameters in different subspaces.

the complete data space. Essentially, parameter importance is infered over much smaller subspaces by utilizing previous coarse grained queries. The main difficulty to this, is that LPMs are able to explain local subspaces. Within those local subspaces the framework utilizes previous AQ to understand parameter importance. If those subspaces do not overlap, then it will be challenging to identify how the AQ behaves within unknown areas. To be more concise, imagine single dimension queries $q \in \mathbb{R}$, where q could be a single input parameter with a true function mapping to results $y \in \mathbb{R}$. Now imagine that q_1, q_2, q_3, are three queries $q_1, q_2, q_3 \in [0, 1]$ such that $q_1 \leq q_2 \leq q_3$. Training our model with those queries leaves the model with high uncertainty as to what happens in the in-between spaces where no training examples are acquired. Trying to approximate the behavior of a *true* function, within those spaces requires good generalizability from the trained models. In our work, this is exactly what we are trying to do. With minimal knowledge given from the coarse grained training examples we are trying to infer the impact of parameters in much smaller unknown spaces. As shown in our experimental section, our models overcome this difficulty and are able to find approximate functions to explain what happens over the smaller subspaces.

3.8.1 Examples of Explanation Functions and Practical Usage

Using an LPM, the analyst obtains the result of queries, not yet executed, and can infer the importance of query parameters under the given subspace. In this section, we provide some examples of how LPMs can be utilized to provide insight and guide analysts during data exploration.

In Figure 3.9 we have created an interactive plot of the different query parameters listed as features and their importance across different clusters using Bokeh [29]. This overview

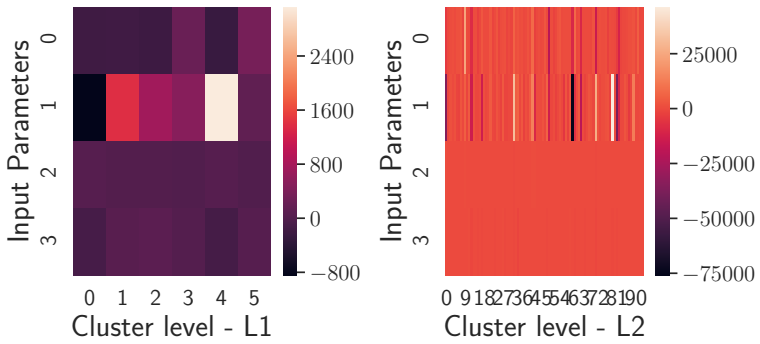

Figure 3.10: Hierarchical representation of L1,L2 clusters and the parameter importance. (Left) Clusters for query centers $\mathbf{x}$ and the average importance of parameters inferred by the underlying LPMs. (Right) An L2 cluster associated with Cluster 0 from L1.

is constructed by averaging the importance of all LPMs associated with L2 clusters and then shown for each L1 cluster. By hovering over a particular rectangle the analyst can see the importance of a parameter over the specified cluster. As evident, the importance of parameters varies across clusters which can lead the analyst to specific subspaces where the parameter is more important. For instance, if a particular parameter has more weight in a given cluster then the analyst knows that varying this parameter will have more impact to the end result than other parameters. In essence, the analyst has a guide to inform them which parameters to shift in certain subspaces such that they get meaningful results and not waste resources executing queries that will not impact the end result.

Moreover, we can construct hierarchical visualizations that show a more fine grained representation of parameter importance. For this example, we used the coefficients of LPMs for explanation functions. Their main difference with parameter importance is that they also indicate in what way they can influence the result as they can be negative or positive. For instance, a large negative coefficient for a specific parameter as seen for input parameter 1 at L1=0 in Figure 3.10 (left) informs us that this particular parameter will have a negative impact on the end query result. In Figure 3.10(left) the average coefficients for all parameters over clusters at L1 are plotted. Using this visualisation, provides an overview of the coefficients associated with each parameter and allows the ability to zoom-in to a particular cluster of L1 and see how the coefficients change at L2 clusters. In this case study, the parameters $0-3$ correspond to:

- 0 : *X Coordinate*
- 1 : *Y Coordinate*
- 2 : *Size of X Coordinate*

- 3 : *Size of Y Coordinate*

These parameters define range queries that cover a spatial region on a map. The corresponding response variable of the LPMs is the *Arrest Rate* within those areas. Therefore, an analyst can start their analysis by consulting this figure. They can initially focus on cluster 0 at L1 which shows the lowest average coefficient for parameter 1 : *Y Coordinate*. The fine grained view shown in Figure 3.10(right) corresponds to the coefficients for all input parameters across all clusters at L1-0. The analyst might then notice that all parameters have positive coefficients except parameter 1 especially over cluster 47 at L2. Given this information, the analysts can obtain the actual LPM model and mean vectors for clusters 0 at L1 and 47 at L2. The mean vectors correspond to the LRs and RRs, together they denote the mean (average) query executed at those regions. Hence, they can identify a possible region to which the arrest rate is smaller than the average arrest rate over other regions *without* executing anything. The mean arrest rate is visualised in Figure 3.11 along with the distribution of arrest rates over the pinpointed region. Using our methodology, the analysts have identified an area with smaller than the mean, arrest rates. By obtaining the coefficients for the corresponding LPM at L1-0 and L2-47, they can observe what drives the arrest rate up or down. The corresponding coefficients in this particular case were $[621.64900659, -455.21564492, 34.93227467, -246.40730294]$ which indicates that as the *Y Coordinate* increases, moving up north on the map, the arrest rate decreases. This is also indicated by the region size covered by the *Y Coordinate*. Using this derived information, the analyst can further investigate as to why that happens. Maybe the number of police officers patrolling the areas has dropped, or maybe the incidents recorded did not warrant an arrest.

Hence, without even executing a single query, the analyst has tremendous knowledge which can guide them when they first initiate their exploratory analysis. Surely, the LPMs used were trained using queries which were executed before. However, these queries can be obtained from past analyses made by other analysts. Therefore, the analysts can save time by focusing on particular areas instead of starting with a coarse grained view with no tool to guide them.

3.9 Experimental Evaluation

We run experiments to evaluate the accuracy and performance/scalability of our proposed solution. Accuracy is evaluated on two axes: (1) Predictive capability of the proposed explanation function; using the function as a surrogate black-box function to compute the result of an unknown query, and (2) Accuracy of estimated explanation functions. We conduct experiments to analyse the performance of our system, measuring the time required for training the different LPMs, and the impact of hyper-parameters in explanation serving.

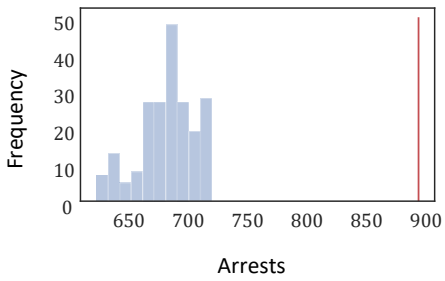

Figure 3.11: The average number of Arrests, across all regions, shown as a red vertical line along with the distribution of arrests for queries executed over the region identified by 0-L1 and 47-L2

3.9.1 Experimental Setup & Metrics

Data Sets and Query Workloads: The real dataset $\mathcal{B}_1 = \{\mathbf{x}_i\}_{i=1}^{N}, \mathbf{x} \in \mathbb{R}^2$ with cardinality $|\mathcal{B}_1| = N = 6 \cdot 10^6$ contains crimes for the city of Chicago [6]. We obtain a synthetic workload $\mathcal{T}$ containing $m = 5 \cdot 10^4$ queries and their answers, i.e., $\{(\mathbf{q}, y)_i\}_{i=1}^{m} = \mathcal{T}$. Each query is a 4-d vector $\mathbf{q} = (\mathbf{x}, \boldsymbol{\theta})$ with answer y where $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ is the center, $\boldsymbol{\theta} \in \mathbb{R}^2$ is its length and $y \in \mathbb{R}$ the result obtained from executing the query against real dataset $\mathcal{B}_1$. We use workload $\mathcal{T}$ for *Pre-Processing* and *Training* and create a *separate* evaluation set $\mathcal{V}$ containing $|\mathcal{V}| = 0.2 \cdot m$ new query-answer pairs. The synthetic query workloads were obtained from [9] and follow a similar generation process as described in [18, 113] also described in [9].

To implement our algorithms, we used `scikit-learn` [101] , `KMeans` [61], and an implementation of the `MARS` algorithm [48]. We performed our experiments on a desktop machine with a Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and 16GB RAM. MARS was used as the explanation function algorithm because of its superior accuracy to simple linear regression. We note that any ML algorithm, which is able to provide an estimate on feature importance to characterize the importance of input parameters is a viable alternative to MARS.

Evaluation Procedure

To evaluate the effectiveness of our proposed explanation functions on an unknown set of AQs, we have to generate perturbed versions of each query $\mathbf{q} \in \mathcal{V}$. Assessing the explanation functions just based on set $\mathcal{V}$ is not enough, as the metrics will report on the accuracy of performing point estimates using the explanation functions. Instead, by perturbing a query vector, we are able to assess whether the proposed explanation function is an accurate approximation of the black-box function generating the results of queries. For a query in $\mathcal{V}$, we

generate a set of perturbations $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ with $n = 100$, where $\mathbf{p} \in [-1, 1]^d$. Using these perturbations, we obtain a set of *sub*-queries $\mathcal{S}$, where $\mathcal{S} = \{\mathbf{q} + \mathbf{p}_1, \ldots, \mathbf{q} + \mathbf{p}_n\}$ and $|\mathcal{S}| = |\mathcal{P}|$. A number of *sub*-query sets are generated which is equal to the number of queries in $\mathcal{V}$, and $\{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{V}|}\}$. The results for each one of those perturbed versions of query $\mathbf{q}$ are computed using the real dataset. Such that, each perturbed version of query $\mathbf{q}$ is also associated with its result $y \in \mathbb{R}$. The explanation function, is then evaluated over each sub-query set $\mathcal{S}_i$ by measuring the loss (EEL) between our predictions and the actual responses. The evaluation metrics are computed for the recorded responses and the responses obtained from the proposed explanation function. We then report on an average of the evaluation metrics.

Evaluation & Performance Metrics

We use a variety of evaluation metrics to measure different aspects of the suggested framework. These metrics can give a holistic overview on the accuracy of an explanation function.

Information Theoretic Metric: The EEL is measured using the Kullback-Leibler divergence (KL). Concretely, the result is a scalar value denoting the amount of *information loss* when one chooses to use the approximated explanation function, for a given unseen query (without executing this query), instead of the actual explanation function (after executing the unseen query). The EEL with KL divergence is defined as:

$$\mathcal{L}(f, \hat{f}) \quad = \quad KL(p(y)||\hat{p}(y)) = \int p(y) \log \frac{p(y)}{\hat{p}(y)} dy, \tag{3.15}$$

with $p(y)$ and $\hat{p}(y)$ being the probability density functions of the true and approximated query result, respectively.

Goodness of Fit: The EEL is measured, using the coefficient of determination R^2. This metric indicates how much of the *variance* generated by f can be explained using the approximated explanation function $\hat{f}$. This represents the goodness-of-fit of an approximated explanation function over the actual one:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}, \tag{3.16}$$

The denominator in (3.16) is proportional to the *variance* of the *true* function and the numerator are the residuals of our approximation. The EEL between f and $\hat{f}$ can be computed as:

$$\mathcal{L}(f, \hat{f}) \quad = \quad 1 - R^2 \tag{3.17}$$

Predictive Accuracy: We also quantify the predictive accuracy associated with explanation (regression) functions. We employ the *Normalized Root Mean Squared Error (NRMSE)* for this purpose:

$$\text{NRMSE} = \frac{1}{y_{max} - y_{min}}(\frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2). \tag{3.18}$$

Essentially, this shows how accurate the results would be if an analyst used the explanation (regression) functions for further data exploration, without executing queries. This is evaluated on set $\mathcal{V}$ as we want to evaluate the accuracy on point estimates.

3.9.2 Experimental Results: Accuracy

For our experiments we chose to show performance and accuracy results over three representative aggregate functions: `COUNT`, `AVG` and `SUM` due to their extensive use in data exploration. We compare the accuracy across the different workloads each one with different distributions for region location and region size. Their abbreviations are shown in Table 3.1. We compare our LPMs with two baselines **Local** and **Global**. The Local model is trained using the perturbed set of queries and answers $\mathcal{S}$, hence, is the gold standard that we are trying to achieve. Please note that obtaining a Local model is not actually possible as one has to execute a number of perturbed versions of a query to be able to obtain it. We are merely constructing such local functions to provide an estimate of the highest possible accuracy that can be obtained. The Global model is obtained by training a single model on $\mathcal{T}$ and is the baseline that we are trying to beat.

	Distribution for $\mathbf{x}$	Distribution for $\boldsymbol{\theta}$
G.G	Gaussian	Gaussian
G.U	Gaussian	Uniform
U.G	Uniform	Gaussian
U.U	Uniform	Uniform

Table 3.1: Abbreviations for the different workloads used

Goodness of Fit Results: Figure 3.12 shows the results for R^2 over all workloads for all three aggregates. We report on the *average* R^2 found by evaluating the models on queries in $\mathcal{V}$ using their pertubed sets $\mathcal{S}$. As expected the Local models perform best as they are trained using queries in the pertubed sets. Overall, we see that our proposed explanation functions approaches the accuracy of the **Local** models and is more accurate than the **Global** function, except for one case in which both the LPM and **Global** achieve similar accuracy. It might be the case that the knowledge obtained from the queries was not enough such that our LPMs

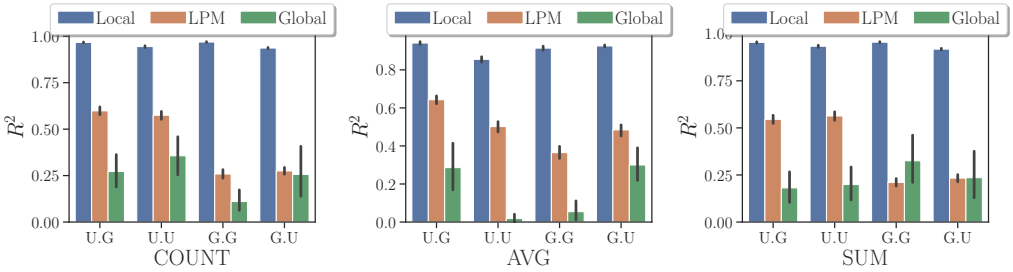

Figure 3.12: Results for R^2.

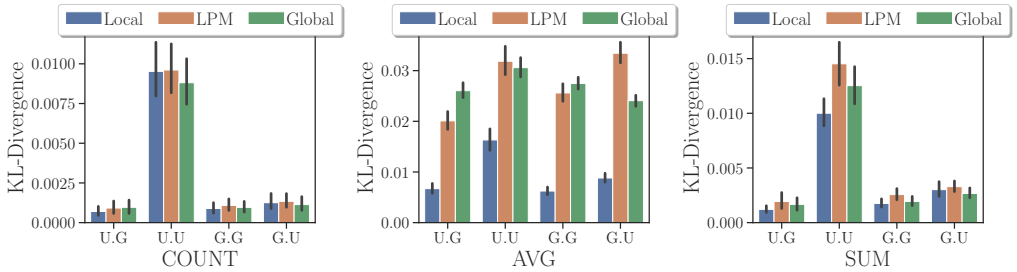

Figure 3.13: Results for KL-Divergence.

would be able to approximate the general trend followed by these queries in more specific subsets. We also notice some difference in accuracy levels across different workloads. This leads us to believe that the distribution of queries is a significant factor in determining the accuracy of the proposed models. However, the general outtake of this experiment is that we can fit good approximate models that approach the accuracy of models fitted with the pertubations themselves.

Information Theoretic Results: Figure 3.13 shows the ratio of bit increase when using the approximated distribution generated by the LPMs rather than the actual values for y of the perturbations (where lower is better). This indicates the inefficiency caused by using the approximated explanations, instead of the *true* function. This information, theoretically, allows us to make a decision on whether using such an approximation, would lead to the propagation of errors further into the data analysis process. We observe, that overall, the ratio is low for all methods with the Local method approximating the true distribution with minimal loss of information. It is important to note that all results are less than 10% with most of the loss incurred over the `AVG` function. We notice that sometimes the **Global** function outperforms our LPMs. However, the difference in all of these cases is minimal. We speculate that the Global model obtains a coarse grained view of how the aggregate statistic across a much larger space which could tend to approximate the underlying distribution for the pertubed queries. These results serve as evidence that using our approximations is sufficient for conducting further analyses, by using these models instead of executing subsequent queries to uncover more information.

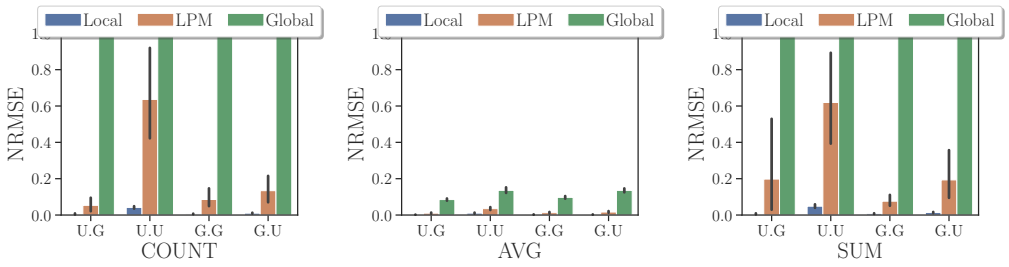

Figure 3.14: Results for NRMSE.

Predictive Accuracy Results: We have also measured the predictive accuracy of all methods, across all datasets to quantify how good the models would be in predicting the results of future queries. The results are shown in Figure 3.14. As expected, the error is the lowest for the Local method as the predictions are being made on the set of queries that the model was trained on. The highest error was recorded by the Global method. Apart from AVG, the Global method has extremely large NRMSE which makes it inappropriate to make predictions for the perturbations, thus, it should not be used in this case. We also note, that across all three aggregates, the U.U distributions appears the hardest to perform predictions. One possible explanation is that models trained on uniform workloads, require more queries as they have to be equally good uniformly across the space that they are being evaluated. Their difference, with workload distributions having a Gaussian distribution, is that queries are executed with a higher probability closer to the mean and thus the patterns might be easier to learn. This experiment also shows that no one model is fit for all problems. So, in this case, it might be more appropriate to investigate different models (with the same properties of interpretability) to be used for different distributions. In conclusion, the accuracy across statistics/aggregates and workloads is not preventive. It shows that our LPMs could be used, in the context of providing approximate answers, to queries that analysts might have while conducting their analysis. This could save up computational/monetary/productivity costs, as the analysts would be using the LPMs to get answers efficiently without issuing queries against *any* of the data.

3.9.3 Experimental Results: Efficiency and Scalability

Training Performance: To effectively measure how long it takes to complete training of LPMs, we restrict the workloads used to "G.G" and set the vigilance thresholds for clustering to the default values of 0.4 for $\boldsymbol{\theta}$ and 0.05 for $\mathbf{x}$. These parameters control the number of clusters created at the L1 and L2 levels. As will be shown in our sensitivity analysis section, the number of clusters follows logarithmic scale with respect to the vigilance parameters. For this experiment, we gradually increase the number of training examples and record the

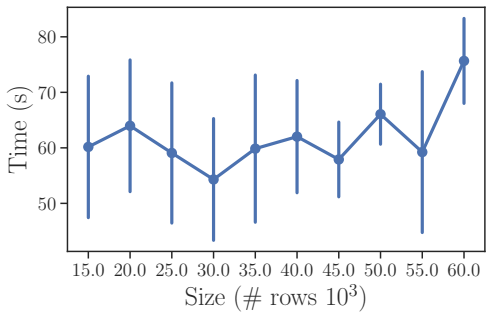

Figure 3.15: Training time for an increasing number of training examples

time (in seconds) required by our framework, to complete training of the LPMs. To account for randomness, we repeat the procedure 10 times per number of training examples. Note that this is part of the Pre-processing step which happens offline and helps to initialize the LPMs. Our framework is also able to adjust its parameters online to account for new queries being executed by the back-end analytics engine.

The results are shown in Figure 3.15, as the number of training examples increases we do not observe a significant increase in the time required to train all LPMs. The time ranges from 40(s) to 80(s) with some significant variation. Further investigation reveals that the number of training examples is not highly correlated with the training time. This is exhibited in Table 3.2, where correlation [1] varies from $[-1, 1]$ with values closer to 1 indicating strong positive correlation and values closer to -1 strong negative correlation. As evident, the number of clusters that formed in L1 and L2 have strong positive correlation with Training Time. This is attributed to the fact that, an increasing number of clusters, means that the hyper-tuning procedure was running for longer. The hyper-tuning procedure was outlined in Algorithm 1.

	L1 clusters	L2 clusters	Training Examples	Training Time (s)
L1 clusters	1.000000	0.990559	-0.018008	0.885539
L2 clusters	0.990559	1.000000	-0.004828	0.886221
Training Examples	-0.018008	-0.004828	1.000000	0.218372
Training Time (s)	0.885539	0.886221	0.218372	1.000000

Table 3.2: Pairwise Pearson's Correlation Coefficient for different parameters.

To reinforce this finding we plot the different runs, as points with a varying color for the number of L1/L2 clusters, shown in Figure 3.16. As the number of training examples (x-axis) increases we see no upward trend (as one would expect), instead if the number of clusters is low the training time for a particular run is at the lower end of the y-axis (Time). However, as the number of L1/L2 clusters increase, Training Times also increase even for runs where we used the smallest number of training examples. This is with an exception for

[1] Pearson's Correlation Coefficient is used.

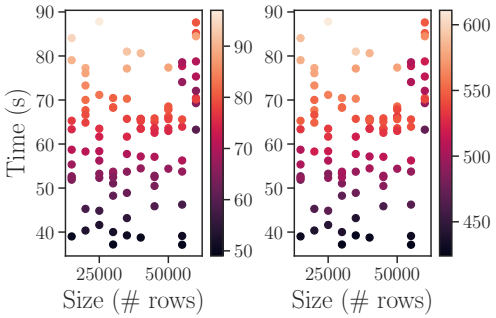

Figure 3.16: Training time for an increasing number of training examples with the number of clusters for (left) L1 and (right) L2 shown as different colors

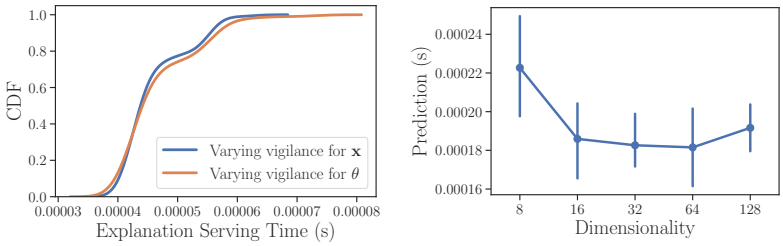

Figure 3.17: (Left) A CDF of the explanation serving time by varying hyper-parameters that control number of clusters for L1/L2. (Right) Prediction serving time plotted against an increasing dimensionality of the input vector $\mathbf{q}$

the largest number of training examples where with an intermediate number of clusters we get Training times at the higher end of y-axis. This goes out to show that even though the number of training examples is not a driving factor it can contribute to the total training time.

Explanation & Prediction Serving Performance: We now assess the Explanation and Prediction serving performance of our framework. The explanation functions and any predictions stemming from the explanation functions (LPMs) have to be returned to the user efficiently. Our goal is to do that in less than 500 ms, as it has been shown that any answer returned over that limit might hinder productivity [85]. For this experiment, we vary the thresholds/vigilance parameters for $\mathbf{x}$ and $\boldsymbol{\theta}$ which control the partitioning levels and hence the number of clusters. The threshold values for both parameters $\mathbf{x}$ and $\boldsymbol{\theta}$ were in the interval of $[0.01, 3)$. Figure 3.17 shows the results of our experiments. Two CDFs are plotted, one where we vary the vigilance parameter for $\mathbf{x}$ with constant vigilance for $\boldsymbol{\theta}$ and vice versa. The creation of more clusters contributes to the total explanation serving time, as the closest clusters (according to our formulation) have to be identified for an LPM to be returned to the user. However, the partitioning level remains constant after exceeding particular threshold values, as the optimal number of clusters is identified (optimal with respect to SSQE). Consulting the CDF in Figure 3.17 we notice that for both cases the explanation serving time is orders of magnitude less than 500ms. In our experiment with varying the threshold for $\mathbf{x}$

we saw no significant impact as the partitioning level quickly reaches the optimal number of clusters (found to be 6). Although we would see an impact in performance if the number of clusters at L1 increase, we note that this is no more than linear as a single pass over the LRs of L1 can tell us which one is closer to the query issued by the analyst. Varying the threshold/vigilance parameter for $\boldsymbol{\theta}$ has more effect as we recorded a mean number of total clusters in L2 of 492 with standard deviation of 211. Still, the explanation serving time is no more than 0.08 ms, $6250\times$ faster than the proposed limit for explanation serving time set at 500ms.

To measure the Prediction serving time we vary the dimensionality of the input vector as this would affect the evaluation of the underlying LPM. We used MARS [48] as our ML model and recorded the time required for making a prediction. We varied the dimensionality within the range $[8, 128]$. An input vector with 128 dimensions means that a query is executed with a multi-dimensional center $\mathbf{x} \in \mathbb{R}^{64}$ which translates to filtering based on 64 different attributes. This might be a bit of an overstretch but we wish to stress test our system and see if LPMs can efficiently provide estimations for the answers of unseen queries. The results are shown in Figure 3.17(right), with the Prediction time fluctuating however never exceeding 1ms. It is important to note that this includes the time of finding the closest LPM.

Overall, throughout our performance experiments we conclude that the use of the proposed framework does not hinder productivity. On the contrary, it efficiently informs the user of the unerlying variation in a statistic using LPMs. The LPMs are located and returned to the user in milliseconds which allows for the analyst to continue ther analysis without bottlenecks. In addition, using LPMs analysts can efficiently get estimates for the answers of any future queries they might have.

3.9.4 Experimental Results: Sensitivity

Threshold/Vigilance Query Center Parameter ($\mathbf{x}$)

Several experiments have been conducted to test the sensitivity of our algorithms to the ϵ threshold used in Algorithm 1. As Algorithm 1 is used twice in the *Pre-Processing mode* we refer to ϵ as the $\mathbf{x}$ Threshold and $\boldsymbol{\theta}$ Threshold to distinguish between the thresholds used in the creation of L1/L2 clustering phases. Effectively, these thresholds control the partitioning level (the number of clusters). As thresholds for parameters $\mathbf{x}$ and $\boldsymbol{\theta}$ become smaller, clustering level is increased. Meaning that parameter K for K-Means is incremented, until the difference of SSQE between partitioning levels of K and $K + 1$ is equal to or below the given threshold value.

We varied the threshold for $\mathbf{x}$ in the interval of $[0.0001, 3)$. The results for this experiment are shown in Figure 3.18. In this figure, we examine the effects of this threshold on both

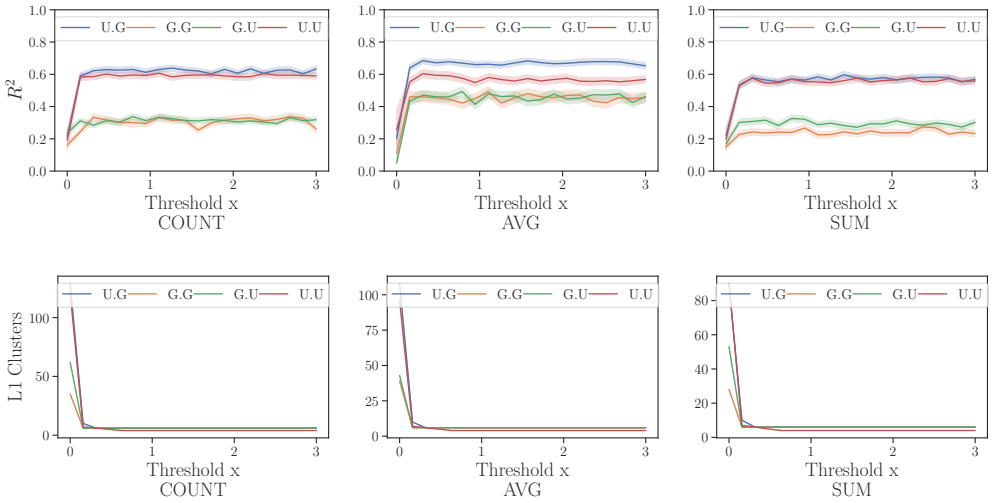

Figure 3.18: Measuring the effects of varying threshold/vigilance parameter for $\mathbf{x}$. (Top) The effects on accuracy R^2 for different workloads and aggregate functions. (Bottom) The effects on the number of L1 clusters for different workloads and aggregate functions.

the accuracy, Figure 3.18(top), and the number of clusters, Figure 3.18(bottom). Firstly, we report that for all workloads and aggregate functions, the effects of this threshold reduce, as the number of clusters reaches the optimal number. We can see that accuracy increases and remains constant as the number of clusters converges. So in short, setting the optimal threshold is important, for achieving high accuracy. In our experiments, we first normalize the data to be at the same scale, which makes setting this threshold parameter easier. In addition, a hyper-tuning procedure could be added for setting this threshold parameter at an optimal value. However this was out of the scope of this work. In general, we can easily determine a near optimal value by randomly testing a number of values along a given range also known as Line-Search.

Threshold/Vigilance Query Parameter (θ)

We also vary the threshold for parameter $\boldsymbol{\theta}$ and investigate its effects on the number of clusters for L2 and accuracy measured by R^2. Its effects are recorded for all workloads and datasets in Figure 3.19. Figure 3.19(top) shows that accuracy is constant as the threshold varies. This means that partitioning the region sizes has less effect than optimally partitioning the query locations $\mathbf{x}$. The same effect is observed across aggregates and workloads. On the other hand, the threshold parameter influences the number of L2 clusters and as the threshold increases the partitioning level becomes more coarse grained and less clusters are created at L2. The number of L2 clusters shown on the y-axis is the total number of clusters associated with all L1 clusters.

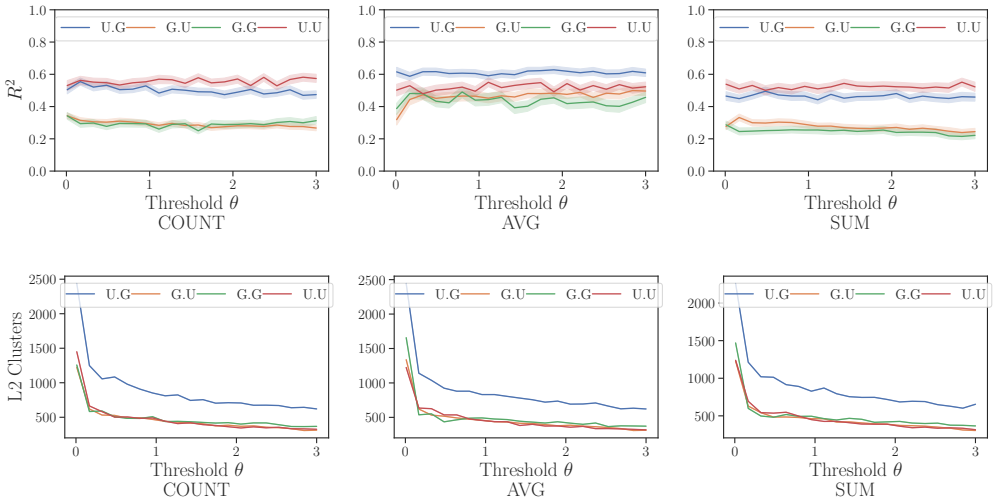

Figure 3.19: Measuring the effects of varying threshold/vigilance parameter for $\boldsymbol{\theta}$. (Top) The effects on accuracy R^2 for different workloads and aggregate functions. (Bottom) The effects on the number of L2 clusters for different workloads and aggregate functions.

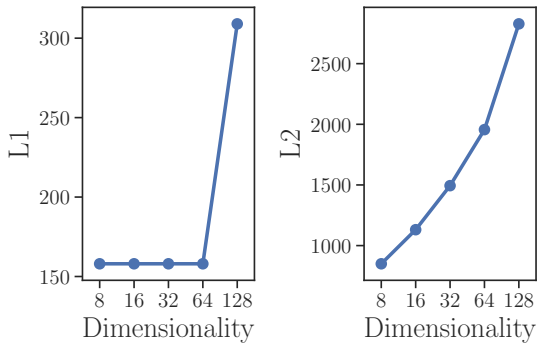

Figure 3.20: As the dimensionality of the query vector increases more clusters are needed both in L1 and L2 to sufficiently reduce SSQE.

Dimensionality of Query Input Vector $\mathbf{q}$

We experiment with an increasing dimensionality for the query/input vector and investigate its effects on the partitioning levels (number of L1/L2 clusters). As evident in Figure 3.20, the number of L1 and L2 clusters increases with respect to dimensionality. There is an exponential increase in the total number of L2 clusters as an increase in L1 clusters has a multiplicative effect on the number of L2 clusters. Hence, dimensionality has an effect on the explanation and prediction serving performance as more clusters have to be investigated.

3.10 Conclusions

In this chapter, we have defined a novel class of explanations for AQs, which are of particular importance for exploratory analytics. The proposed AQ explanations are succinct, presented in the form of regression functions. They convey rich information to analysts about the queried data subspaces and explain how an aggregate value depends on key parameters of the queried space. Furthermore, they allow analysts to utilize these explanation functions for data exploration without the need to issue more queries to the DBMS. This is because the proposed explanation functions can be used to estimate the result of an AQ given the parameter values used in an AQ. We have formulated the problem of deriving AQ explanations as a joint optimization problem and have provided novel learning routines for its solution. Specifically, the joint optimization problem is deconstructed into three parts which we solve separately. The proposed scheme for computing explanations does not require DBMS data accesses ensuring efficiency and scalability as we utilize QDL for all phases of our methodology. In addition, we have shown examples of how the explanation functions can be leveraged by analysts when performing data exploration. Data analysts, can consult the visualisations, constructed using the explanation functions, to identify subspaces of interest a-priori without executing any query against the underlying backend system. Overall, this chapter focused on presenting alternative uses of the QDL mechanism by showing how past queries can be utilized, to construct explanations that can assist analysts during data exploration. We believe this is a significant step forward in expediting the data analytic process without human supervision/intervention.

Chapter 4

Identifying interesting subspaces with Query-Driven Surrogate Models

4.1 Introduction

Consider Exploratory Data Analysis (EDA), whereby analysts engage in repeatedly selecting regions in their data and subsequently summarizing them by extracting statistics [66]. For instance, analyzing spatial data one might filter out all data points except the ones of a specific district and then measure the number of data points within that region to infer the interestingness of it. Multiple methods/algorithms/visualizations implicitly adopt this process and are part of an analyst's toolbox. A challenge to this approach is that the task of mining regions of interest is a tedious and laborious process and in the worst case has exponential complexity. The interestingness of a region can be measured by comparing its extracted statistic with a cut-off value or a given threshold by the analysts/applications. Regions to which the extracted statistics are greater/less than a given threshold are deemed more interesting. This approach is found in numerous applications. For instance, in cluster analysis [109], this approach is used to decide which clusters to prune. In addition, similar approaches are adopted in detecting regions of interest in fMRI scans [103] (where only the regions that are '*activated*' are shown). Lastly, regions are exluded by a given threshold in the process of identifying landmarks [132], based on tracking data. For these reasons, we believe mining regions that exceed a statistic threshold is in many cases more appropriate than having the analyst ask for the $top - k$ regions of interest.

4.1.1 Use Case Examples

Let 2-dimensional spatial coordinates describe the locations of Crime Incidents (or any data points with spatial dimensions like traffic congestion and pollution levels in urban areas,

etc.). Proactively identifying regions which contain a pre-defined number of data points within them can advise analysts as to which areas are worth exploring. For instance, a region having more Crime Incidents than a global threshold or having higher (local) average deprivation/crime-index indicator could suggest lack of infrastructure, policing or social/economic disparities compared to other regions. Identifying such regions is not trivial and we will show that a naive approach has exponential complexity.

The use cases are not restricted to density of data points within regions. For instance, using data from activity trackers, analysts wish to find time-frames (regions in time) with high ratio of a specific activity (e.g., sitting, standing etc). Those time-frames comprise crucial information to the activity patterns of a user. If other attributes are also incorporated, like GPS coordinates, geo-spatial readings from accelerometers, the analysts can be advised as to when & where an activity occurs most often, along with what type of readings indicate the activity is taking place. Note that the regions of interest denote boundaries in multidimensional space thus making them easy to interpret.

Moving to high dimensional use cases, within ML classification problems, analysts are often interested in finding regions with high ratio of certain classes (class-labels), thus, implicitly suggesting classification boundaries. This task cannot be performed visually unless dimensionality reduction is employed which does not guarantee fine-grained and accurate results (going back to original space) and may suggest regions which are no longer interpretable.

4.1.2 Contributions

In this chapter, we describe a methodology to reduce the complexity of searching for regions of interest *per* analyst request given a threshold. We continue in the same vein of Chapter 3, in that we focus on alternative use cases of QDL, which are directed towards automating the data exploration processes. We formulate the problem at hand as an optimization one which can be of multimodal nature as multiple regions matching the analyst request can exist. We identify the back-end data/analytics system as being a bottleneck in examining the validity of the proposed regions. To alleviate this key problem, we propose the use of QDL to approximate the behavior of the back-end system, i.e., to find surrogate models that replace the back-end data system for this task. We then use these models, produced by QDL, in an evolutionary multimodal optimization (based on the principles of swarm intelligence) to identify the regions of interest per analyst request. We name the model proposed in this paper as SuRF, after the terms **Su**rrogate **R**egion **F**inder. To this end, this chapter's contributions can be summarized by the following points:

- The task of mining interesting regions is formalized. The formulation is based on objective functions that incorporate statistics and a user-defined cut-off value.

- How QDL and multimodal swarm optimization algorithm can be used to locate multiple regions of interest.
- The use of QDL to approximate the back-end system. In a similar manner as previous chapters, we exhibit how QDL can be used along with evolutionary optimization to alleviate the inherent complexity of the considered task.
- Finally, we provide extensive experimental results evaluating and comparing the proposed approach and the various algorithmic strategies with other methods.

The rest of the chapter is organized as follows: Section 4.2 formalizes the problem of finding regions of interest and describes a baseline algorithm, which is of exponential complexity. Section 4.3 defines the optimization problem at hand and introduces evolutionary multimodal optimization for solving it. Section 4.4 describes the type of surrogate ML model needed to approximate the behavior of the back-end analytics system. Finally, Section 4.5 contains a comprehensive list of experiments and results that assess the accuracy and efficiency of SuRF.

4.2 Problem Definition & Rationale

In this section, we initially provide some definitions that will be helpful throughout this chapter. The challenge of identifying interesting regions is then formalized and a baseline algorithm is provided. We show that the complexity of such an algorithm is exponential which makes it unattractive for this task.

Definition 4.2.1. (*Data Vector*) Let $\mathbf{a} = (a_1, \ldots, a_d)^\top \in \mathbb{R}^d$ denote a multivariate random data vector. A dataset $\mathcal{B}$ is a collection of N data vectors $\{\mathbf{a}_k\}_{k=1}^N$.

Definition 4.2.2. *(Statistic Region)* We define a *statistic region* in a d-dimensional vector space via the $(2d+1)$-dimensional information vector $\mathbf{q} = (\mathbf{x}, \mathbf{l}, y)^\top$, where $\mathbf{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$ is the region center point of the hyper-rectangle with side lengths $\mathbf{l} = (l_1, \ldots, l_d)^\top \in \mathbb{R}_+^d$ across the d dimensions. A statistic region $\mathbf{q}$ over dataset $\mathcal{B}$ is associated with the subset $\mathcal{D} \subseteq \mathcal{B}$ encompassing vectors $\mathbf{a}$ such that $\{\mathbf{a} \in \mathcal{D} | \bigwedge_{i=1}^d (x_i - l_i \leq a_i \leq x_i + l_i)\}$. A statistic region is similarly defined as AQs in prior chapters. However, we refer to them as statistic regions as SuRF is not necessarily tied to any DBMS, but is instead a generic method that can be used within any data analytic environment.

The component $y = f(\mathbf{x}, \mathbf{l})$ denotes a mapping $f : \mathbb{R}^d \times \mathbb{R}_+^d \mapsto \mathbb{R}$ over $\mathcal{D}$ from the hyper-rectangle $(\mathbf{x}, \mathbf{l})$ to a statistic of interest $y \in \mathbb{R}$, i.e., scalar y is the *statistic* extracted from the data vectors in $\mathcal{D}$. This can be (not limited to) e.g., number of vectors in $\mathcal{D}$, i.e., $y = f(\mathbf{x}, \mathbf{l}) = |\mathcal{D}|$, or the average $\bar{a}_i$ of dimension a_i, i.e., $y = f(\mathbf{x}, \mathbf{l}; i) = \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} a_{i,k}$, $\mathbf{a}_k \in \mathcal{D}$.

Note that in the case of the average $\bar{a}_i$ the i-th dimension is not part of the defined hyper-rectangle and the definition becomes $\{\mathbf{a} \in \mathcal{D} | \bigwedge_{j_{-i} \in d}(x_j - l_j \leq a_j \leq x_j + l_j)\}$.

Definition 4.2.3. *(Surrogate Model)* Given a region $\mathbf{q}$, the corresponding mapping f returns a *local* statistic of interest. Function f depends on the conditional data distribution $p(\mathbf{a}|\mathbf{x}, \mathbf{l})$ defined by the hyper-rectangle $(\mathbf{x}, \mathbf{l})$. The actual evaluation of f is computationally expensive as one has to identify the complete data subset $\mathcal{D}$ given region $\mathbf{q}$ out of all data points. Therefore, we rest on an approximate surrogate model $\hat{f}$ to approximate f, i.e., $f \approx \hat{f}$ given any random $\mathbf{q}$. Such approximation exploits past actual evaluations of f given random regions. Using QDL, we obtain a surrogate model $\hat{f}$, thus, we can avoid an expensive evaluation of f, given a random region $\mathbf{q}$. We can instead evaluate $\hat{f}$ that can approximate the results given by f. This yields orders of magnitude speed-ups with a trade-off in accuracy, since the evaluation of $\hat{f}$ does not involve identification and access to the data vectors in $\mathcal{D}$.

Problem 1. Given a user requested cut-off value $y_{\text{R}} \in \mathbb{R}$, seek the k unknown regions $\{\mathbf{q}_k\}$ over the vectorial space of $\mathcal{B}$ such that their corresponding statistics $\{f(\mathbf{x}_k, \mathbf{l}_k) = y_k\}$ are less (or greater) than y_{R}. That is, find the k unknown regions $\{\mathbf{q}_k\}$ defined by $(\mathbf{x}_k, \mathbf{l}_k)$:

$$\{\mathbf{q}_k \in \mathbb{R}^{2d+1} : f(\mathbf{x}_k, \mathbf{l}_k) < y_{\text{R}}, \forall k\}. \tag{4.1}$$

Note: we adopt $(y_k > y_{\text{R}})$ in the case where the sought statistics are all greater than y_{R}. For instance, find the areas where the crime index is greater than 60%, i.e., the statistic here is the number of crime incidents of areas. Alternatively, the task could be to identify areas where the average deprivation score is less than the expected one.

To avoid the inherent computationally heavy task of evaluating all possible (not trivially countable) sub-regions that satisfy (4.1) (see later), we approach a solution to Problem 1 using surrogate models $\{\hat{f}\}$ over a dataset $\mathcal{B}$. Evidently, this approach introduces approximation of the evaluation of (4.1) by *replacing* $f(\mathbf{x}_k, \mathbf{l}_k)$ with $\hat{f}(\mathbf{x}_k, \mathbf{l}_k)$. We also, re-write Problem 1 to be expressed as an optimization problem. We define an objective function that helps us find multiple regions by finding *local*-optima. In the optimization function, we incorporate region *size* defined by $(\mathbf{x}, \mathbf{l})$, which is of high importance as an arbitrarily large size might not be informative enough. For instance, if we seek regions with population number larger than y_{R} with $y_{\text{R}} < |\mathcal{B}|$, then a region covering all data vectors (whole data-space) is the optimal result. Therefore, we factor in the region size in the objective function defined as:

$$J(\mathbf{x}, \mathbf{l}) \quad = \quad \frac{y_{\text{R}} - f(\mathbf{x}, \mathbf{l})}{\left(\prod_{i=1}^{d} l_i\right)^c}. \tag{4.2}$$

The objective illustrated in (4.2), indicates the consideration of the total area covered by the sought region in the denominator. A single *global* optimal solution maximizing the objectve in (4.2) would be an infinitesimal box surrounding a single point with the greatest difference given by $y_{\text{R}} - f(\mathbf{x}, \mathbf{l})$. Indeed, this would be a valid solution and might be of interest to the analyst. However, as we will later show there could be multiple *local* optimal solutions meeting the constraints (introduced in (4.3)) and maximizing (4.2). Hence, we are not interested in finding one global solution to the given objective, but many. For this reason, we introduce a tuning scalar parameter c, which allows the user to restrict to smaller/larger areas. Hence, we seek the region(s):

$$(\mathbf{x}^*, \mathbf{l}^*) = \arg \max_{(\mathbf{x},\mathbf{l}) \in \mathbb{R}^{2d}} J(\mathbf{x}, \mathbf{l}) \quad \text{s.t. } f(\mathbf{x}, \mathbf{l}) < y_{\text{R}}. \tag{4.3}$$

In the case of the constraint being $f(\mathbf{x}, \mathbf{l}) > y_{\text{R}}$, we maximize $-J(\mathbf{x}, \mathbf{l})$. In the remainder we use (4.3) without loss of generality to either case. To avoid extra computational complexity we take the logarithm of (4.2) obtaining:

$$\mathcal{J}(\mathbf{x}, \mathbf{l}) = \log(J(\mathbf{x}, \mathbf{l})) = \log(y_{\text{R}} - f(\mathbf{x}, \mathbf{l})) - c\|\boldsymbol{\xi}\|_1, \tag{4.4}$$

where $\boldsymbol{\xi} = [\log(l_1), \ldots, \log(l_d)]^\top$ and $\|\boldsymbol{\xi}\|_1 = \sum_{i=1}^{d} \log(l_i)$ is the L_1 norm of the log-vector of $\mathbf{l} = [l_1, \ldots, l_d]^\top$. An interesting property arises from (4.4) as the logarithm is undefined for negative values. Thus, the objective implicitly rejects regions in which $y_{\text{R}} - f(\mathbf{x}, \mathbf{l}) < 0$ conforming to the constraint of finding regions less than y_{R} (and vice versa for $f(\mathbf{x}, \mathbf{l}) > y_{\text{R}}$), as will be shown in our experiments. In (4.4), $c > 0$ is the L_1 regularization parameter limiting the size of $\boldsymbol{\xi}$ (and of $\mathbf{l}$) coefficients and results in finding fine-grained regions (in size), as discussed later.

4.2.1 Baseline Complexity

Before elaborating on our computationally efficient solution for Problem 1, we first describe a baseline solution. The computational complexity of mining k regions in (4.1) grows exponentially with data dimensionality d. It is not trivial to find exact solutions given continuous data domain of (if not all) different dimensions in $\mathcal{B}$. Given continuous (real-values) attributes x_i, one way of solving Problem 1 is to perform an exhaustive search. Initially, data is discretized using a finite number of multidimensional center points to obtain n regions $\{\mathbf{x}_1 \preceq \mathbf{x}_2 \ldots \preceq \mathbf{x}_n\}; \mathbf{x}_i \in \mathbb{R}^d$ ($\preceq$ denotes the point-wise inequality between values of the same dimension). This discretization yields an approximate solution, as the optimal center for a region could lie in-between the proposed centers. In addition, the arbitrary size of the regions adds another level of complexity to the exhaustive search as we have to consider n regions with varying sizes across dimensions, such that $\{\mathbf{l}_1 \preceq \mathbf{l}_2, \ldots \preceq \mathbf{l}_m\}$, which again is

an approximate size of the optimal region. Thus, to obtain potential regions via exhaustive search yields asymptotic complexity of $\mathcal{O}((n \times m)^d)$. We then have to evaluate the result for each of the obtained regions using (4.4). Since the objective in (4.4) entails the evaluation of f over $\mathcal{D} \subseteq \mathcal{B}$, the baseline complexity becomes $\mathcal{O}((n \times m)^d \times N)$, assuming that f can be computed in a single pass over $\mathcal{B}$ in linear time. As dimensions d and data vectors N grow, the task becomes prohibitively costly. In the next sections, we discuss how to leverage evolutionary multi-modal optimization algorithms and surrogate models to reduce the complexity associated with an exhaustive search.

4.3 Optimization & Viable Solutions

Given that the baseline complexity of solving this task becomes exponential we seek alternative solutions. Our task is to maximize the objective in (4.4) in an efficient manner. We first discuss about the form of the objective which will help us identify candidate optimization algorithms.

The solution space of the objective in (4.4) might have a unique (optimum) or multiple solutions (local optima) given an arbitrary y_{R} by the user. Based on Problem 1, given a y_{R}, the probability of finding a *viable* region is

$$\mathbb{P}\{f(\mathbf{x}, \mathbf{l}) > y_{\text{R}}\} \quad = \quad 1 - F_Y(y_{\text{R}}), \tag{4.5}$$

where F_Y is the CDF of y. Since, $\lim_{y_R \to +\infty} F_Y(y_R) = 1$, it indicates that the objective function will have less viable solutions because $\mathbb{P}\{f(\mathbf{x}, \mathbf{l}) > y_{\text{R}}\} \to 0$, i.e., the probability of a viable solution diminishes. In the case $f(\mathbf{x}, \mathbf{l}) < y_{\text{R}}$, we obtain $\lim_{y_R \to -\infty} \mathbb{P}\{f(\mathbf{x}, \mathbf{l}) < y_{\text{R}}\} = \lim_{y_{\text{R}} \to -\infty} F_Y(y_{\text{R}}) = 0$. Hence, with an *appropriate* y_{R}, i.e., strictly non-zero probability (4.5), we expect to find multiple regions (local optimal) satisfying (4.1), i.e., $k \geq 1$ regions. It is highly plausible that given an *appropriate* y_{R}, multiple regions k exist satisfying $f(\mathbf{x}_k, \mathbf{l}_k) > y_{\text{R}}$. Therefore we make use of a *multimodal optimization* algorithm capable of finding all the possible solutions for Problem 1.

4.3.1 Multimodal Evolutionary Optimization for Regions Finding

Due to the multimodal nature of Problem 1, we cannot adopt optimization methods which return a single optimal solution (=region) given y_{R}. Therefore, we cast the optimization problem as an evolutionary multimodal optimization problem [139, 79] adopting methodologies from Swarm Intelligence. We adopt the Glowworm Swarm Optimization (GSO), which is a multimodal variant of the well-known Particle Swarm Optimization (PSO) method [72].

Both GSO and PSO methods are computationally light, providing near-optimal solutions (regions in our context) in the face of non-differentiable *fitness* objective functions. Notably, GSO optimizes multimodal fitness functions as it converges towards multiple local-optima, thus considered a good candidate optimizer for our problem.

GSO makes use of *particles*, which are represented as multidimensional candidate solutions in the solution (region) space. It adopts a mechanism to move those particles around the solution space, which converge eventually to local-optima. A candidate solution particle $\mathbf{p} = (\mathbf{x}, \mathbf{l}) \in \mathbb{R}^{2d}$ refers to a region defined by $(\mathbf{x}, \mathbf{l})$ in the $(2d)$-dimensional solution space. The fitness objective function that we use for GSO is the objective $\mathcal{J}$ in (4.4), which encapsulates function f. However, given an arbitrary y_{R}, our method avoids the evaluation of f over all possible viable solutions. The fitness function of GSO becomes the objective $\hat{\mathcal{J}}$ derived from (4.4) by replacing f with the estimate $\hat{f}$. Hence, the solutions are evaluated using $\hat{\mathcal{J}}$ given estimate $\hat{f}$.

In short, GSO initializes a number of particles $\{\mathbf{p}_i\}$ at random positions in $\mathbb{R}^{2d}$. Each particle $\mathbf{p}_i$ is associated with a *luciferin* value ℓ_i emulating glowworms. The GSO algorithm is an iterative algorithm, with discrete steps $t = \{1, 2, \ldots\}$ and is split into two phases. The first phase updates the luciferin $\ell_i(t)$ at step t for each particle $\mathbf{p}_i = (\mathbf{x}_i, \mathbf{l}_i)$ in the swarm using:

$$\ell_i(t) = (1 - \rho)\ell_i(t - 1) + \gamma\hat{\mathcal{J}}(\mathbf{x}_i, \mathbf{l}_i) \tag{4.6}$$

The factor ρ in (4.6) is the luciferin decay, which reduces attraction to particles that are not moving towards local-optima. The factor γ in (4.6) is the luciferin enhancement and increases attraction of particles close to local-optima dictated by the current evaluation of $\hat{\mathcal{J}}$. The second phase, updates the (position) vector $\mathbf{p}_i$ of each particle with respect to a neighbourhood of particles $\mathcal{N}_i(t) = \{\mathbf{p}_j : \|\mathbf{p}_i - \mathbf{p}_j\|_2 \leq r_i(t) \wedge \ell_j(t) > \ell_i(t)\}$ in which the selected neighbours have higher luciferin values and are within a current radius $r_i(t)$ in L_2 (Euclidean) distance. GSO then adapts the (position) vector $\mathbf{p}_i$ towards a neighbour $\mathbf{p}_j \in \mathcal{N}_i(t)$ with the maximum selection probability:

$$\mathbb{P}\{\mathbf{p}_j\} \;=\; \frac{\ell_j(t) - \ell_i(t)}{\sum_{k \in \mathcal{N}_i(t)} \ell_k(t) - \ell_i(t)} \tag{4.7}$$

Fig. 4.1 illustrates the final (converged) positions of the particles over a 2-dim. region space. The x-axis denotes the center of a region, x, and the y-axis denotes the side length l. Hence, each particle is a region defined over this space, with the intensity of the color at Figure 4.1 being the value of the objective function (4.4) across the space. The final positions are illustrated as red "$\times$" and the slightly shaded blue dots are previous positions held by those particles. In this example, 84% of the particles have converged to regions satisfying the constraint, set here at $(f(\mathbf{x}, \mathbf{l}) > 1080)$, $y_R = 1080$. As witnessed, a large number of

particles have converged to the objective's peaks which suggest better regions. Indeed, the regions at the bottom (the peaks) constitute pre-defined *ground-truth* regions (explained in section 4.5 of this chapter). There are also particles that seem stationary as they are in a space *undefined* by our objective (4.4), where ($f(\mathbf{x}, \mathbf{l}) < 1080$). We also explain this in more detail in section 4.5.

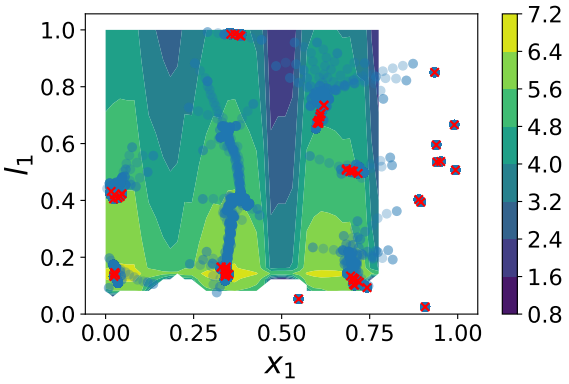

Figure 4.1: Final positions of particles (optimal regions) in the 2-dim. region solution space. The objective's (4.4) value is the color's intensity with the peaks shown at the bottom of the figure. The white color in the plot corresponds to areas where the objective's value is *undefined*, meaning that a solution does not exist for these parameter values.

4.3.2 Constraining the Regions Solution Space

We introduce the use of surrogate models to expedite the process of evaluating viable regions. However, the surrogate models are not restricted within a specific domain. Although, the underlying f is undefined in regions with no data points in $\mathcal{B}$, surrogate models are not. The purpose of ML models, is to generalize to unknown regions. Hence, even if the function f is undefined in areas with no data points, $\hat{f}$ will still return a result. If the surrogate model is not provided with training examples denoting where the function is undefined then the obtained result might not reflect reality. Therefore, we have to adapt our algorithm to account for this fact.

In addition, the particles in GSO are initially randomly spread across the solution space. The valid solution space (space where data points and thus regions exist) is not reflected and particles only have their neighbours' luciferin values to guide them. These are inherently associated with the fitness value $\hat{\mathcal{J}}$, which then goes back to the initial concern about the validity of the surrogate models. To alleviate this, we first approximate the distribution of the data points $p_A(\mathbf{a})$ (over a sample for large-scale datasets) in $\mathcal{B}$ adopting Kernel Density Estimation (KDE) [124] and then we obtain the probability of a region containing any number of

data points, i.e., from $\mathbf{x}-\mathbf{l}$ to $\mathbf{x}+\mathbf{l}$. We use this as a guide for particles when selecting which direction to explore. Therefore, given (4.7), we alternate the selection probability by multiplying with the density (probability) of data points around the particle $\mathbf{p}_j$'s data component $\mathbf{x}_j$:

$$\mathbb{P}'\{\mathbf{p}_j\} \quad = \quad \frac{\mathbb{P}\{\mathbf{p}_j\} \cdot \int_{\mathbf{x}_j-\mathbf{l}_j}^{\mathbf{x}_j+\mathbf{l}_j} p_A(\mathbf{a})d\mathbf{a}}{\sum_{k\in\mathcal{N}_i} \mathbb{P}\{\mathbf{p}_k\} \cdot \int_{\mathbf{x}_k-\mathbf{l}_k}^{\mathbf{x}_k+\mathbf{l}_k} p_A(\mathbf{a})d\mathbf{a}} \tag{4.8}$$

4.3.3 Complexity of Multimodal Optimization

As reported earlier, the baseline complexity of the presented challenge is $\mathcal{O}((n \times m)^d \times N)$. By adopting GSO and surrogate models $\hat{f}$, we expedite this process, obtaining viable solution(s) in $\mathcal{O}(TL^2d)$, where T is the number of iterations and L is the number of particles for GSO. As a rule of thumb, GSO requires less than $T \approx 100$ iterations and $L \approx 100$ glowworms to converge (empirically demonstrated in section 4.5). On the contrary, using the baseline/naive approach with just $n = m = 6$ and $d = 5$, one needs to evaluate more than $6 \cdot 10^7$ possible regions over N data points. On the other hand, GSO has to execute only $100 \times 100 = 10^4$ evaluations, just 0.016% of the evaluations needed by the baseline approach.[1] Therefore, by using GSO, the complexity is now of polynomial nature as not all parameter values, spanning uniformly across the entire domain space, have to be examined. In addition, the use of surrogate models and QDL, has eliminated the need to examine N data points as the regions no longer have to be evaluated using f. In the next section, we report on how to approximate f using ML models. This gives a near-constant time (with respect to the chosen model) performance for evaluating the region's statistic y.

4.4 Surrogate Model Estimate

We could approximate f via various ML models[2] trained to associate a region $(\mathbf{x}, \mathbf{l})$ with its corresponding statistic $y = f(\mathbf{x}, \mathbf{l})$ using a set of past function f evaluations in $\mathcal{Q} = \{\mathbf{q}_m = (\mathbf{x}_m, \mathbf{l}_m, y_m)\}_{m=1}^M$ inline with QDL methodology that was presented so far. Using these *training* examples, ML models approximate the actual f. In general, ML algorithms try to minimize the EPE $\min_{\hat{f}} \mathbb{E}[(f(\mathbf{x}, \mathbf{l}) - \hat{f}(\mathbf{x}, \mathbf{l}))^2]$ which is estimated using an out-of-sample dataset different from $\mathcal{Q}$. They also try to find models which are complex enough to minimize this EPE and simple enough to ensure good generalizability to never before seen

[1] **Note:** Although the complexity contains $T \times L^2$, the number of region evaluations by the algorithm is, in fact, $T \times L$ [79].

[2] We restrict to a single class of ML models in our experimentation, however this is not necessary and alternative ML models could be employed.

examples: they tune what is called the Bias-Variance trade-off to ensure the derived model is neither under-fitting nor over-fitting [46]. However, our task is to approximate the behavior of the actual f applied over regions of data subsets in $\mathcal{B}$. Hence, our primary concern is *not* to generalize well to new examples. Instead, it is to find a surrogate model $\hat{f}$, which follows the *trend* of f over random regions given an arbitrary y_{R}. In other words, our desideratum when identifying $\hat{f}$ is that given a random region $(\mathbf{x}, \mathbf{l})$, if the statistic $y = f(\mathbf{x}, \mathbf{l})$ and $f(\mathbf{x}, \mathbf{l}) < y_{\mathrm{R}}$ then (and only then) the estimate $\hat{y} = \hat{f}(\mathbf{x}, \mathbf{l})$, and $\hat{f}(\mathbf{x}, \mathbf{l}) < y_{\mathrm{R}}$. That is both f and $\hat{f}$ should *agree* on the constraint $< y_{\mathrm{R}}$ for any random region. This, clearly by definition, does not imply that $|y - \hat{y}|$ is desired to be as small as possible (i.e., minimizing the prediction error). Instead, we would like to obtain a model $\hat{f}$ such that whenever $y < y_{\mathrm{R}}$ holds then, $\hat{y} < y_{\mathrm{R}}$ holds true, too. Surely, if $\hat{f}$ minimizes the EPE then we may statistically expect that the two above-mentioned conditions hold true. Nonetheless, both conditions can hold true even if it is not the case that $y \approx \hat{y}$. To reflect this objective, we would require to find an estimate $\hat{f}$, which minimizes the L_2 norm difference of gradients at any region:

$$\min_{\hat{f}} \mathbb{E}[\|\nabla \hat{f} - \nabla f\|_2] \tag{4.9}$$

Minimizing the gradient difference we expect that a surrogate model $\hat{f}$ resembles the behavior of the true underlying function f. However, a number of problems arise if we seek to minimize (4.9). We have no way of knowing if the true function f is differentiable and we also do not restrict our choice of ML models to differentiable ones. We could approximate the gradient using a finite number of training samples that are equally spaced in $(\mathbf{x}, \mathbf{l})$. But this would mean that we cannot take advantage of past function evaluations, issued by analysts/applications, as an assumption that these examples are equally spaced is invalid.

In this paper, we do not use a specific class of ML models that minimizes (4.9) and is left as our future work for further investigation. Nevertheless, we adopt conventional ML models minimizing the EPE, which can be directly used for providing robust (in terms of predictability) surrogate estimate model $\hat{f}$.

4.5 Performance Evaluation

In our evaluation, we seek to answer the following:

1. What is the impact on accuracy, for finding interesting regions per user request using QDL and surrogate models $\hat{f}$?

2. What are the performance benefits of SuRF over the baseline approach and other methods?

3. How sensitive is SuRF to the choice of hyper-parameters and how is efficiency and accuracy affected by different choices of objective functions?

We begin by outlining the implementation details & setup, discussing our methodology and establish evaluation metrics in Section 4.5.1. We showcase the accuracy of SuRF in comparison to other methods using a variety of synthetic datasets in Section 4.5.2. A qualitative analysis over real datasets, showing the applicability of SuRF is presented in Section 4.5.3. The performance benefits of SuRF are discussed in Section 4.5.4. The aforementioned sections provide the answers to questions (1) and (2). Finally, we answer question (3) by evaluating the sensitivity of objective functions, GSO and surrogate ML models in Sections 4.5.6, 4.5.7, and 4.5.8, respectively.

4.5.1 Implementation Details & Setup

We implemented our algorithms using scikit-learn [102] and used XGBoost (XGB) [35] ML model for our surrogate models $\hat{f}$. We implemented GlowWorm [79] as our optimization algorithm. We performed our experiments using Python 3.5 running on a desktop machine with an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and 16GB RAM. The surrogate models used for both synthetic and real datasets were trained using a set of past function evaluations executed across the data space with centers $\mathbf{x}$ selected uniformly at random and region side lengths $\mathbf{l}$ set to cover $1\% - 15\%$ (uniformly) of the data domain.[3] The surrogate models were hyper-tuned using Grid-Search [102] with K-fold cross validation. A sensitivity analysis for surrogate models is discussed at Section 4.5.8. **Note:** A Github repository was created to help aid the reproducibility of our experiments at `https://github.com/Skeftical/SuRF-Reproducibility`.

Methods: We evaluate the effectiveness and efficiency on mining interesting regions of four different methods:

1. SuRF : Our framework SuRF which is the surrogate model, trained using QDL, and used with the GSO algorithm.
2. `Naive` : The baseline method described in Section 4.2.1[4]
3. f+GlowWorm : The GSO optimization coupled with the true underlying function which accesses data to evaluate the objective function described in (4.4)

[3] Please note that uniformly sampling regions across the data space with uniform lengths is not the same as obtaining training examples that are *equally spaced* across the complete domain in both $\mathbf{x}$ and $\mathbf{l}$

[4] As the number of function evaluations becomes un-manageable we restrict the discretisation to $n = m = 6$

4. PRIM : An implementation of the algorithm described in [50] and its implementation is obtained from [4]. PRIM is used to find regions which maximize the result of an output variable. We have found it performs good on our task as well.

Synthetic Datasets: We have created 20 synthetic datasets to compare the methods outlined above. We use synthetic datasets as there are currently no real datasets that provide a list of interesting regions a-priori. Hence, there is no way to verify whether algorithms are indeed accurately identifying interesting regions. In addition, the use of synthetic datasets gives us the flexiblity to adjust controlling parameters and observe the effect they have on the evaluated models. The size of the datasets can be arbitrary and it is defined within each experiment. The synthetic datasets have Ground Truth (GT) regions, which are purposely either more dense than the rest of the dataset, or have relatively higher y values (for the purposes of testing for other statistics). The GT regions are hyper-rectangles constraining a region in all dimensions. Concretely we vary the following settings:

- Number of GT regions : $k = \{1, 3\}$.
- Statistic type for y : Either: (i) 'density' referring to number of data points in subset $\mathcal{D}$ or (ii) 'aggregate' referring to average value of a certain dimension of data points in subset $\mathcal{D}$
- Data dimensions : $d \in \{1, 2, 3, 4, 5\}$.

Each dataset is characterized by a variation of these settings. Note that the statistic could be any other type, e.g., variance, high-order moments. Figure 4.2 shows four different datasets with varying settings. The sub-figures on the left column, show data points $\mathbf{a}$ when data dimension is set to $d = 1$, as only the dimension a_1 is to be used to bound the data space. The dimension a_1 has areas with larger values for a_i and thus the average $y = \frac{1}{|\mathcal{D}|} \sum_{m=1}^{|\mathcal{D}|} a_{i,m}$ over the highlighted GT regions bounded on a_1 is higher. On the other hand, the sub-figures on the right column, show the corresponding datasets for the 'density' statistic. The region is bounded by both a_1 and a_2 and for the highlighted (green rectangle) GT area, the density of data points is higher. The number of GT regions $k = 3$ is evident at the bottom sub-figures, in which multiple regions exist for both statistics, and $k = 1$ at the top sub-figures.

Our goal, for each synthetic dataset, is to estimate the GT boundaries as close as possible. Let $\mathcal{R}(\mathbf{x}, \mathbf{l})$ be the hyper-rectangle area corresponding to a random region $(\mathbf{x}, \mathbf{l}) \in \mathbb{R}^{2d}$ with coordinates: $(\mathbf{x} - \mathbf{l}, \mathbf{x} + \mathbf{l})$. We use a popular metric adopted in data mining, the Intersection over Union (IoU), also known as the *Jaccard Index*. Which is a ratio where the numerator is the area of overlap between the bounding box (hyper-rectangle) $\mathcal{R}(\mathbf{x}_k, \mathbf{l}_k)$, mined from any of the outlined methods, and the GT bounding box $\mathcal{G}(\mathbf{x}_0, \mathbf{l}_0)$ corresponding to the GT

region $[\mathbf{x}_0, \mathbf{l}_0]$. The denominator is the area of union, i.e., the area encompassed by both the $\mathcal{R}(\mathbf{x}_k, \mathbf{l}_k)$ and the GT bounding box $\mathcal{G}(\mathbf{x}_0, \mathbf{l}_0)$, thus, we obtain:

$$\text{IoU} = \frac{\mathcal{R}(\mathbf{x}_k, \mathbf{l}_k) \cap \mathcal{G}(\mathbf{x}_0, \mathbf{l}_0)}{\mathcal{R}(\mathbf{x}_k, \mathbf{l}_k) \cup \mathcal{G}(\mathbf{x}_0, \mathbf{l}_0)}, \tag{4.10}$$

where $\cap$ and $\cup$ in (4.10) are adopted as the overlap and union operators over (hyper)-rectangles. One might notice that region dimensionality is not exceedingly high (we experiment up to $2d = 10$ dimensions in the region solution space for $d = 5$ data dimensionality). Indeed, at first we conducted experiments by producing synthetic datasets $\mathcal{U}(0, 1)^d, d \gg 5$, resulting to searching for regions in significantly higher than 10-dimensional spaces. However, due to the effects of curse of dimensionality and as mentioned by Friedman et al. [46], regions (and data points) become increasingly sparse and, thus, the mined regions were returning no data points, thus, no *interesting regions*. The expected length l of a hyper-cube to retrieve a fraction , r, of data points in unit volume in $\mathbb{R}^d$ is given by $\mathbb{E}[l; r] = r^{\frac{1}{d}}$ [46] (Section 2.5 of [46]). Thus, as dimensionality d increases, the expected length becomes much larger, covering most of the data domain. Hence, the notion of finding interesting regions becomes meaningless as we would essentially return regions covering most of the data domain. Even though we set the synthetic datasets' dimensionality up to 5, we highlight the fact that our algorithm deals with $2d$ dimensions as our regions are expressed as vectors in $\mathbb{R}^{2d}$ (region solution space).

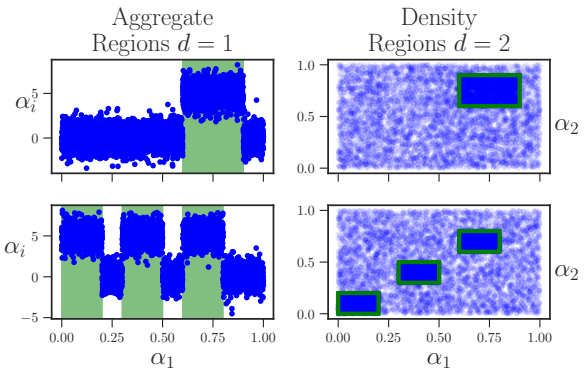

Figure 4.2: Synthetic Ground Truth Regions (shaded green) for statistic type 'aggregate' and $d = 1$(left) and ground truth regions (green rectangles) for statistic type 'density' and $d = 2$ (right), with both a single ground truth region $k = 1$ (top) and multiple regions $k = 3$ (bottom).

Real Datasets: We use the `Crimes` [6] and `Human Activity` datasets [22] publicly available online. As ground-truth regions do not exist for these datasets, we use them to conduct a qualitative analysis experiment, testing the applicability and effectiveness of SuRF to find regions of interest for fixed y_{R}. Specifically, we train surrogate models using function

evaluations obtained uniformly across the data space with varying lengths and, then, try to find regions of interest given y_{R}. Finally, we analyze the obtained regions and confirm that they match to true regions in those datasets. Parameter c for objective (4.2) was set to $c = 4$ in these experiments.

4.5.2 Accuracy of Interesting Region Identification

All experiments, for assessing the accuracy of interesting region identification were performed on the constraint $f(\mathbf{x}, \mathbf{l}) > y_R$ with y_R set to the value of the extracted statistic given by the GT regions. Specifically $y_R = 2$ for aggregate statistics and $y_R = 1000$ for the *density* statistic. As stated, the surrogate models were trained using past function evaluations, the number of past function evaluations varied as the number of dimensions increases $(300 - 300\mathrm{K})$ to account for the fact that more training examples are required to sufficiently learn a much larger space. The GSO parameters were dynamically adjusted to reach convergence outlined in Section 4.5.7. The objective's parameter was set to $c = 4$. For PRIM, minimum support for the sub-boxes was set to 0.01 and the threshold for aggregate statistics to 2. For `Naive`, as the number of queries becomes prohibitively large, we resort to a subset of the total queries that are to be generated. Nevertheless, this is still a good approximation for the method outlined in Section (4.2.1) and serves as a good baseline. As the synthetic dataset size in this experiment is not important we create synthetic datasets of $7,500 - 12,500$ points. Bigger datasets will merely scale the responses. For all algorithms, we obtain the average IoU per dataset by obtaining all the proposed regions given by the algorithms and assessing their IoU with the GT regions.

Figure 4.3 shows the average IoU over all settings used. As dimensionality increases, the IoU decreases for all methods across all settings. It is worth mentioning that our method is identical to the true underlying function method (f+GlowWorm) without incurring any of the costs associated with computing the exact results of the statistics. This leads us to believe that the error attributed to the use of an approximation is minimal and, thus, it can be safely used to identify interesting regions with no significant use of computational resources. From all sub-figures, we can deduce that dimensionality plays a crucial role in making this task more challenging. We see a drop in IoU as $d > 3$. One contributing factor, is that the GT regions cover a much smaller space in higher dimensions. Given a fixed side length of $l = 0.3$ in uniform space $\mathcal{U}(0, 1)$, then the ratio of space covered in $d = 1$ can be obtained by $0.3^d = 0.3^1$. As d increases then the ratio of space covered becomes much less and thus the probability of fully intersecting with other hyper-rectangles is relatively small. For instance, the ratio of covered space (by the GT) in $d = 3$ is 2.7% of the total space covered by the unit hyper-cube.

For the aggregate statistic and $k = 1$ (top-left sub-figure of Figure 4.3), PRIM outperforms

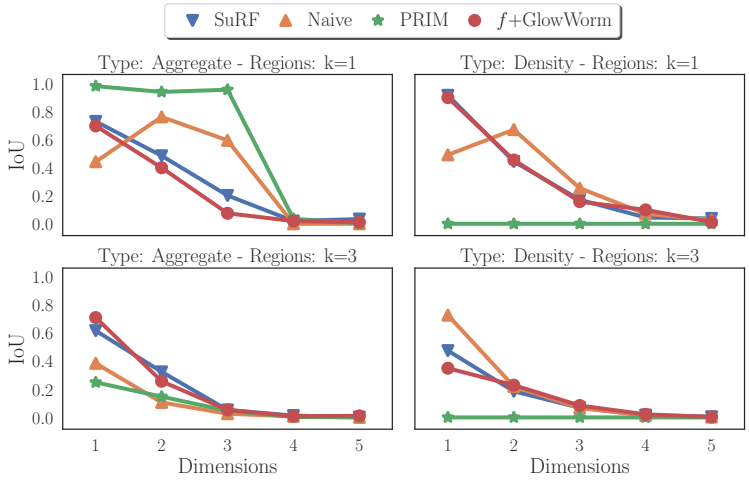

Figure 4.3: Average IoU: (Top-Left) for aggregate statistic and $k = 1$ GT region; (Top-Right) for density statistic and $k = 1$ GT region; (Bottom-Left) for aggregate statistic and $k = 3$ GT regions; (Bottom-Right) for density statistic and $k = 3$ GT regions.

all other methods and is initially invariant by the increase in dimensions. However, for the density statistic (right column in Figure 4.3), PRIM is unable to spot the GT regions as it is not applicable in such domains. PRIM constructs sub-boxes (hyper-rectangles) by *peeling* across a specific dimension. It sequentially generates smaller sub-boxes B until the support of current box β_B (i.e., $\beta_B = |B|$; the number of points belonging in B) is below a user-specified threshold β_0. PRIM tries to identify sub-boxes with minimum support β_0, that maximize the average response value of a selected attribute. Formally, PRIM's objective is:

$$\max_{B} \mathbb{E}[f(\mathbf{a})|\mathbf{a} \in B \wedge \beta_B = \beta_0]. \tag{4.11}$$

The density of a box B is defined by the support to volume ratio: $\frac{|B|}{\prod_i^d l_i}$, where the denominator is the volume of the sub-box. To this end, there is neither a way to specify *density* as the response variable, nor PRIM takes into consideration the volume of sub-boxes. In addition, PRIM progressively removes sub-boxes such that the expectation in (4.11) is greater than what it was before the removal of the sub-box. In case where two sub-boxes B_i and B_j provide similar gains with respect to (4.11), then the one with *less* support $\beta_{B_i} < \beta_{B_j}$ is removed. However, in the case of the density statistic and, precisely because PRIM does not consider the region covered by the sub-boxes, a sub-box with higher density might be removed. Specifically, consider that there exist two boxes that both maximize (4.11) and also have the same gain; then it might be the case that $\beta_{B_i} > \beta_{B_j}$ and $\frac{|B_i|}{\prod_k^d l_k} < \frac{|B_j|}{\prod_k^d l_k}$. Which means that even if the support of a sub-box is smaller, its density might be larger. Of course this should not be considered as a disadvantage of PRIM, as we are testing the algorithm in settings that was not designed to operate. Its primary use case is to maximize the average

Method	CoV
SuRF	1.11
`Naive`	1.10
PRIM	2
f+GlowWorm	1.26

Table 4.1: Coefficient of Variation for IoU across methods.

response of an attribute by enclosing small sub-boxes in d-dimensional space.

PRIM also performed less than the rest methods for the aggregate statistic and $k = 3$ multiple regions (bottom-left) in Figure 4.3) [5]. In general, we are able to get satisfactory IoU with the `Naive` method, but as we will exhibit in our performance section, its efficiency deteriorates as datasets grow in size and dimension.

Figure 4.4 shows the average IoU along with the standard deviation for multiple/single regions (left) and different statistic/aggregate types (right). For multiple regions, Figure 4.4(left) we note that PRIM has relatively largest standard deviation and largest decrease in accuracy as we switch from 1 GT regions to 3; this can also be assessed by their associated Coefficient of Variation (CoV) at Table 4.1, i.e., the ratio of the standard deviation to the mean. Given that, the lower the value of CoV, the more precise the estimate becomes and the fact that PRIM has the highest CoV, it indicates that it is unstable, across various settings. The other methods have similar CoV's across settings.

In addition, all other methods seem to be identical, with a decrease experienced from 1 GT region to 3 GT regions. On the other hand, the statistic type (density or aggregate) in Figure 4.4(right) does not affect accuracy, apart for PRIM's, which as stated is not able to find regions under this setting. Given our experiments, it is safe to conclude that SuRF is able to detect multiple regions of interest under different types of statistics.

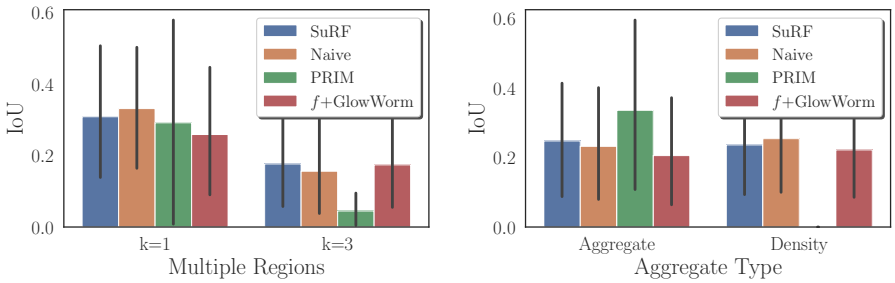

Figure 4.4: (Left) Average IoU for multiple regions; (right) Average IoU for different statistics.

[5] The IoU for $k = 3$ is obtained by averaging IoU's for 3 GT regions.

4.5.3 Qualitative Analysis over Real Datasets

We also run a set of experiments over real datasets to exemplify the use cases of SuRF. Using the approach described, we examine whether SuRF can indeed identify regions of interest experimenting with `Crimes` [6] and `Human Activity` [22] real datasets. SuRF was trained using synthetically generated past region evaluations. We use SuRF over `Crimes` to identify regions where the crime index is over the 3rd quartile of a random set of regions, i.e., $y_R = Q_3$ with $\hat{f}(\mathbf{x}, \mathbf{l}) > y_R$. Figure 4.5 shows the number of crimes over X-Y spatial coordinates. The higher the intensity of the color, the higher the crime rate is within the given area. We plot the corresponding density values obtained by the surrogate model $\hat{f}(\cdot)$, shown in Figure 4.5(left), and note that it is a coarse grained approximation to the true density values shown on the right. However, optimizing the objective function using the surrogate model is still sufficient to propose accurate regions in a matter of seconds. The regions shown in Figure 4.5(left) are the regions that SuRF identified as complying with the constraint $\hat{f}(\mathbf{x}, \mathbf{l}) > y_R$. Figure 4.5(right), shows the same regions over the true density values with 100% of the proposed regions complying with $f(\mathbf{x}, \mathbf{l}) > y_R$. This means that the obtained region defined by $(\mathbf{x}, \mathbf{l})$ complied with the constraint $> y_R$ at both the surrogate $\hat{f}$ and the true function f. Thus, SuRF using approximate surrogate models and GSO is able to pin-point regions of interest in the true data space, complying with the user request $f(\mathbf{x}, \mathbf{l}) > y_R, y_R = Q_3$. Moreover, the regions identified are highly parsimonious as the regions denote boundaries in X-Y Coordinates.

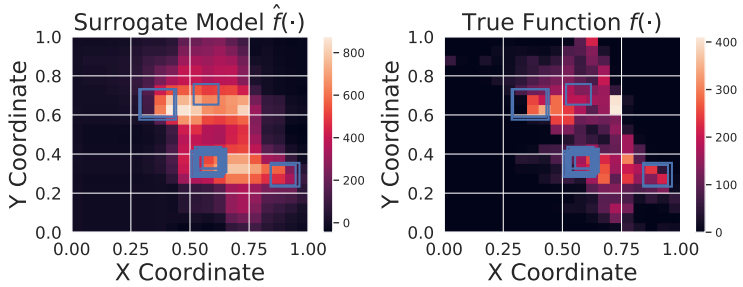

Figure 4.5: On the left, identified regions by approximate surrogate function $\hat{f}$ match to regions identified by the true function f shown to the right.

Furthermore, the `Human Activity` dataset reports the values for gyrometers and accelerometers. Using the parameters `(X, Y, Z)` from the accelerometers, we used SuRF to identify regions with high ratio for a specific activity; for this experiment we used the human activity `stand`. This proactively suggests classification boundaries which the analysts can adopt to build a baseline classifier, or further investigate the identified region. SuRF was able to identify regions with ratio of 33% for activity `stand`. Notably, the empirical CDF $\hat{F}_Y$, where Y corresponds to the ratio of data points with activity=`stand`, showed that the

Table 4.2: Comparative Assessment of Different Methods.

Method	**Data size** N d **dim.**	10^5	10^6 **Time (sec)**	10^7
SuRF	1	1.28	1.28	1.3
	2	1.4	1.4	1.4
	3	1.35	1.35	1.35
	4	1.63	1.63	1.64
	5	1.68	1.68	1.69
`Naive`	1	0.01	0.16	1.94
	2	3.22	33.72	341.7
	3	115.49	1221.6	- (22%)
	4	- (66%)	- (6%)	- (0.5%)
	5	- (1%)	- (0.1%)	- (0.01%)
f+GlowWorm	1	4.71	51.9	601.32
	2	26.7	280.14	2856.02
	3	26.46	289.5	2808.42
	4	27.1	293.62	2981.81
	5	30.21	320.03	-
PRIM	1	0.15	0.4	4.8
	2	0.2	1.9	32.2
	3	0.56	9.3	46.3
	4	0.9	9.5	160.5
	5	1.28	7.36	282.6

probability of obtaining $y_R = 0.3$ was equal to $\mathbb{P}(f(\mathbf{x}, \mathbf{l}) > y_R) = 1 - \hat{F}_Y(0.3) = 0.0035$. This denotes a highly unlikely event and also shows that regions with higher ratios are not easy to identify. This denotes the capability of SuRF to mine interesting regions even for cases where the users' requests correspond to highly unlikely regions.

4.5.4 Models Comparison

We present a comparative assessment with other methods to showcase the efficiency and scalability of SuRF in terms of data size and dimensionality. We also demonstrate the exponential complexity of the considered problem. The performance results are shown in Table 4.2. As shown in Table 4.2, the `Naive` method is efficient with low dimensional data ($d = 1$). For `Naive`, we kept $m = n = 6$, therefore the number of function evaluations executed were just $(6 \times 6)^1 = 36$ for $d = 1$. However, there is an exponential increase in time as d increases, and with $N = 10^7$ data points, `Naive` times out. The ratio included denotes the number of regions examined before exceeding the time limit, which was set to 3000 seconds. The same trend appears in f+GlowWorm showing an increase in the amount of time it takes to mine interesting regions. The GSO parameters were set to $T = 100$ and

$L = 100$ for both f+GlowWorm and SuRF, with initial swarm neighborhood range $r_0 = 3$ and constants $\gamma = 0.6, \rho = 0.4$ as in [79]. For these experiments, we keep GSO's parameters fixed to explore the effects of dimensionality d and data size N. At (4.5.7) we investigate the impact of GSO's parameters on efficiency. PRIM is not affected as much and performs well across all configurations except when the dimensions d and data points N become sufficiently large. On the other hand, SuRF only takes a few seconds across all configurations. Given the same dimesionality d and a varying dataset size N, SuRF's performance remains constant (scales very well) as SuRF does not actually access any data during the mining process. Of course, SuRF's surrogate models are trained before hand for separate statistics. The models will be trained once on a number of past region evaluations and then successively be used for different statistics, thresholds and by different users. Each new request does not need to re-train the model and the overhead for training the surrogate models of SuRF is incurred once. **Note:** It is worth mentioning that all datasets were loaded in memory for performing these experiments. For larger datasets in size N that do not fit in memory the methods in comparison would have to perform multiple disk accesses, thus, incurring significantly higher costs in solving the discussed mining task. In addition, as stated in [50], PRIM is not equipped to work with disk-access and a common remedy would be to sample the dataset. On the contrary, SuRF models are light enough, to always be loaded in memory and make no use of data at all. Hence, for SuRF, it does not matter if the data are stored in disk or on a remote data center.

4.5.5 Training Surrogate Models

In this experiment, we measure the overhead required to train surrogate models on a varying number of queries. The results are shown in Figure 4.6. Using GridSearchCV [102], we are able to find optimal parameters for our model of choice. For GridSearchCV, we pre-specify a range of parameter values for the parameters of XGBoost. We hypertune the parameters:

- `learning_rate` $\in [0.1, 0.01, 0.001]$
- `max_depth` $\in [3, 5, 7, 9]$
- `n_estimators` $\in [100, 200, 300]$
- `reg_lambda` $\in [1, 0.1, 0.01, 0.001]$

As expected, this takes more time than only training the models with their values pre-specified, as witnessed in Figure 4.6. This is because $3 \times 4 \times 3 \times 4 = 144$ combinations have to be tested on large sets of training examples. We could possibly reduce the number of parameter values, to be tested, to increase efficiency. However, we run the risk of not

getting adequate approximations to f. Surely, this should not be a problem to the analysts as the models will only be trained once. In addition, the models could be trained in a central location on more powerful clusters to expedite this process and then subsequently be used by the analysts.

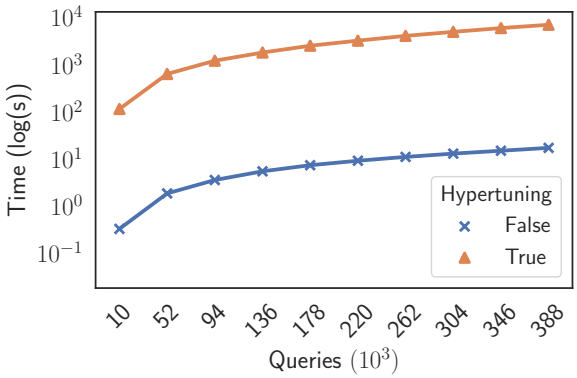

Figure 4.6: Training overhead shown in log-scale (y-axis) as the number of queries (x-axis) increase.

4.5.6 Comparison of the Optimization Functions

We compare the effectiveness of the optimization objectives outlined in (4.2) and (4.4) and present the results in Figure 4.7. The top sub-figures refer to the objective function in (4.4) and the bottom sub-figures refer to objective function in (4.2). We used the synthetic dataset with $d = 1$ and $k = 3$ to be able to visualise the objectives and demonstrate the multimodality in the optimization proccess. In all sub-figures, as the region-size optimization parameter c increases, we observe much more contracted peaks. This is because viable region lengths, l_1',s are restricted to smaller values. Regarding the objective (4.4), the use of logarithms, forces regions not adhering to the constraint on y_R, to become invalid and the corresponding objective function to be undefined. Hence, the white area in Figure 4.7(top) corresponds to those areas. Using objective (4.4), GSO is able to successfully *isolate* glowworms initialized in those areas and eventually adjust their radii to reach glowworms in the valid solution space only. On the other hand, if objective (4.2) was to be adopted, the glowworms could have formed neighbourhoods in what they would believe are local optima, where in reality, those regions would be invalid. In addition, we compare the obtained average IoU and computational performance of the objectives shown in Table 4.3 using the same dataset with $d - 1$ and $k = 3$. Although we do not notice vast differences in IoU, which means that using both objective functions SuRF is able to mine interesting regions with the same accuracy, the performance benefits of using objective (4.4) are clear, as it is computationally cheaper by isolating non-viable region solution sub-spaces and because of its structure.

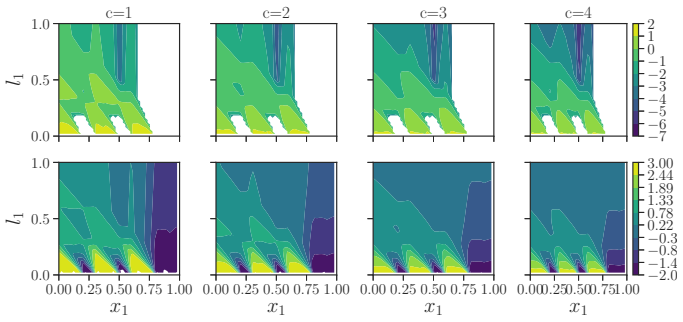

Figure 4.7: 2-dim. region solution space examined by (top) objective $\mathcal{J}$ in (4.4) and by (bottom) objective J in (4.2) as the optimization parameter c increases.

Table 4.3: Performance & Accuracy Comparison of Different Objectives

Method	**Time** (sec)	Average **IoU**
Objective $\mathcal{J}$ in (4.4)	8.81 ± 0.001	0.74 ± 0.08
Objective J in (4.2)	16.3 ± 11.8	0.77 ± 0.03

4.5.7 SuRF-GSO Algorithm Sensitivity

We have conducted experiments to evaluate the computational efficiency of GSO and also examined its rate of convergence across different dimensions and parameter settings. Please note that the *Dimensions* parameter has been doubled to reflect the fact that SuRF (and GSO) operate at $2d$ dimensions per our definition for regions at Def.4.2.2. GSO specific parameters such as γ, ρ are constant adopted from the respective paper [79]. The results are shown in Figures 4.8 & 4.9. Experimentally, we have found that the number of glowworms and neighbourhood radius (r_0 in GSO parameters) have to be adjusted to account for the enlarged region solution space. We increase glowworms using $L = 50d$ and radius $r_0 = (1 - \frac{1}{2}\frac{1}{L})^{\frac{1}{d}}$ adopted from [46] Section 2, Equation (2.24). Although the number of needed iterations does vary across settings, as witnessed in Figure 4.8, the average number of iterations across all settings is 63. Hence, GSO is a robust and efficient algorithm, converging to the various local-optima of the mining task, over different dimensions d and multiple regions k.

Moreover, the average performance for varying number of iterations and glowworms is shown in Figure 4.9. In Figure 4.9(left), we increase dimensionality d and number of glowworms L as we keep the number of iterations $T = 100$ constant to measure the impact on performance for a varying number of glowworms. This has minimal effect on the total run-time as it takes no more than 15 seconds for GSO's process to complete. The same holds for the number of iterations in Figure 4.9(right). Although the average number of iterations required to reach convergence is estimated to be 63 and no setting required more than $T = 250$ iterations, we measured the performance for up to $T = 400$ iterations with

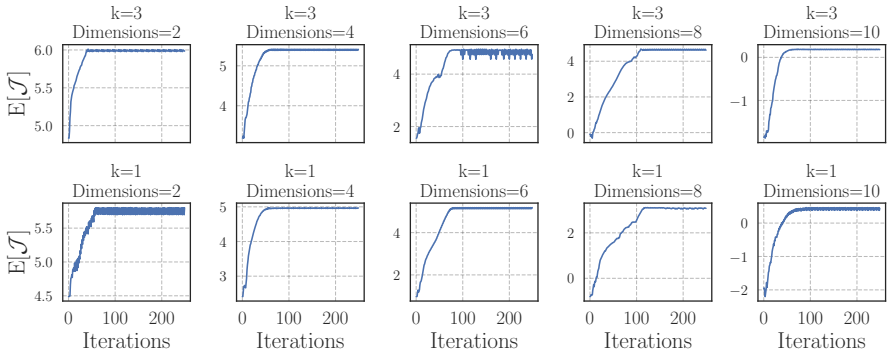

Figure 4.8: Expected convergence rates vs iterations T for different dimensionality d with $k \in \{1, 3\}$ multiple regions.

$L = 100$. No more than 10 seconds is required for the largest number of iterations to finish. It appears that both parameters cause a super-linear increase in time for the same number of dimensions even if the stated complexity was $\mathcal{O}(TL^2d)$. This is because the number of glowworms is small enough so that the time required is still driven by the prediction time from the approximate $\hat{f}(\mathbf{x}, \mathbf{l})$ instead of the increase in glowworms.

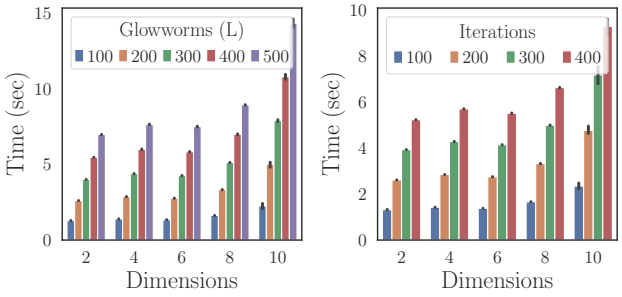

Figure 4.9: SuRF-GSO mining performance over dimensionality d for (left) different number of glowworms L and (right) iterations T

4.5.8 SuRF-Surrogate Model Sensitivity

In this experiment, we evaluate the sensitivity of the surrogate models. Specifically we examine how the number of training examples and out-of-sample generalization error affect the accuracy of the model. In addition, we evaluate how the complexity of the model affects the accuracy of the model and its ability to obtain good IoU.

Figure 4.10 (left) shows a negative correlation between IoU and Root Mean Squared Error (RMSE) obtained from the surrogate models using XGB. For this experiment we use the dataset with a *density* static, dimensions $d = 3$ and single GT region $k = 1$. As the out-

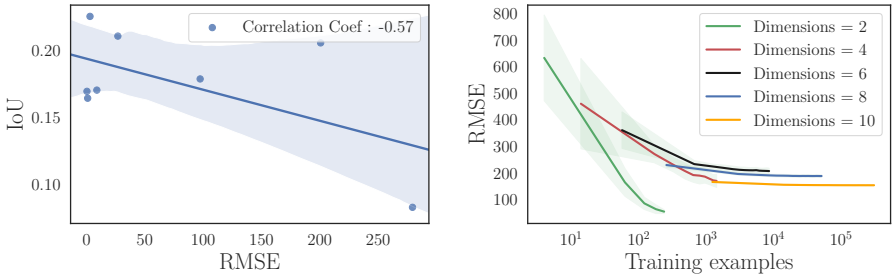

Figure 4.10: (Left) Correlation of IoU and RMSE; (right) number of training examples needed to minimize RMSE of XGB ML-approximate surrogate model over different dimensionality d.

of-sample test error (measured by RMSE) increases, the accuracy for IoU drops. This is evidenced by an estimated regression line along with 95% confidence interval and Pearson's Correlation estimated at -0.57. Therefore, it is important to find ML models that can also act as good statistic estimators. In addition, Figure 4.10 (right) shows how cross-validated error decreases as the number of training examples for approximating a surrogate function increases. For each ML model at different dimensions, we stop training when no further improvement is measured with respect to RMSE. We use datasets with varying dimensions using the *density* statistic and single region $k = 1$. The shaded area refer to the error's standard deviation. We note that by $\sim 1,000$ training examples, i.e., function evaluations, and sufficient hyper-tuning of parameters, the ML models are able to learn the association between region vectors $(\mathbf{x}, \mathbf{l})$ and statistic values y well enough. In our region identification accuracy experiments, we examine the IoU behaviour up to 5-dim. hyper-rectangles corresponding to 10-dim. vectors. Hence, the ML models need to learn using $2 \times d$-dim. vectors. The number of examples is not at all hard to obtain as in reality multi-dimensional regions are extracted from datasets by a plethora of business intelligence applications. One could also assume that the past function evaluations can be obtained, manually by SuRF, at regular downtimes of the system (where traffic load is low).

We also analyze the impact of the XGB-ML model complexity on RMSE and IoU reflected by the maximum depth in regression trees in XGB. The results for both training and cross-validation steps are shown in Figure 4.11. As expected, RMSE drops as ML model complexity is increased. Although not initially evident, IoU has a tendency to increase when model complexity increases. However, this finding suggests, that we could safely omit training a complicated model, as it is evident that they would be able to get a good enough approximation with relatively less complex models. ola

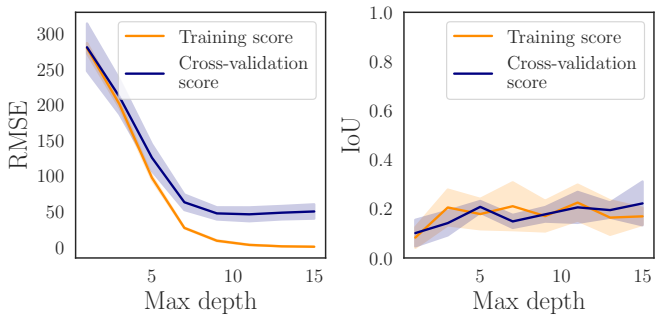

Figure 4.11: (Left) RMSE vs ML model complexity (max depth in trees in XGB); (right) IoU vs ML model complexity (max depth in trees in XGB).

4.6 Related Work

Identifying interesting regions can be traced back to Friedman et al. [50] who were interested in finding regions in d-dimensional spaces that would maximize/minimize a dependent variable y. Their algorithm processed data sequentially to generate smaller regions (as met in regression trees) and included a pruning step in the end. The computational cost of their algorithm is prohibitive when considering large datasets with respect to dimensionality and number of points. In addition, the objective described in [50], is different than the one described in this chapter. SuRF does not seek regions that would maximize/minimize y but regions that satisfy the conditions listed at (4.3). Moreover, our task is loosely coupled with the objective of Subspace Clustering (SC) [100]. The algorithms proposed for SC aim to identify clusters in low-dimensional sub-spaces, by pruning regions and dimensions using some interestingness criterion. The interestingness criterion is often the support (e.g., number of data points with respect to total number of data points) of a given region. Other measures of interestingness have also been proposed, [120] with the underlying metric still being the number of points. Such methods rely on partitioning/discretising schemes, evaluating the density of the found regions and pruning/merging until converging to a region of interest. This is not ideal as the complexity is often exponential with respect to the number of dimensions [120] as also mentioned and experimentally evidenced by our Naive/Baseline solution. In addition, although we consider the density of regions as one example use case, in general, we are interested in regions satisfying the constraint outlined in (4.3) for any given statistic. Thus, our objective is substantially different than the one described in SC work, but we regard it as equally important for data mining practitioners.

Furthermore, there is large body of work on Subgroup Discovery (SD) [24, 56, 28], of course the list is not exhaustive. The purpose of SD is to find subsets of data that show an interesting behaviour with respect to a given interestingness/quality function. It is similar to SC, however SD generalizes the notion of interestingness to subsets of data (potentially across all

dimensions) of various data types, i.e nominal, binary, numeric etc. Depending on the data type, an analogous quality/interestingness function is employed. Multiple algorithms, both exhaustive [25] and approximate [24] have been developed for this task. However, to our knowledge most algorithms are data-driven and do not share our approach to this problem. By data-driven we mean that they employ algorithms that work directly with the underlying data and try to extract subgroups by repeatedly performing region evaluations. This approach is costly as datasets become larger. On the other hand, by using QDL, SuRF can scale regardless of dataset size, as it identifies interesting regions without accessing any of the data.

In other contexts, finding interesting regions was explored for categorical attributes by the construction of OLAP cubes [110] on defined dimensions and hierarchies. As we consider the problem of identifying regions in continuous attributes this approach could not be leveraged as also mentioned in [110], in which they direct to other techniques for continuous data. Alternative formulations, such as posing this problem as finding the top-k regions [87], could also be leveraged and are considered to be complementary to our approach. In the case of, *top-k* formulation, the user has to supply the number of regions, this is often ad-hoc and as evidenced by our examples at Section 4.1 a threshold is more intuitive. Hence, each approach can be used in cases when one of the values (k or threshold) is known. In addition, the complexity of any top-k algorithm inevitably depends on N (the number of data items), d, and k. In intended applications, N will be very large and (as we argued before, so will be k). In contrast, our approach manages to offer performance independent of N, which is likely to pay off big dividends for big data deployments. Also, note that for the multi-modal case in our experiments, if all top-k regions were to be concentrated in one of the GT regions (if y was to be slightly higher for one of the regions) then a top-k approach would effectively identify just one of the regions.

Lastly, spatial indexing [58, 87] could also be considered as the indexes produce hyper-rectangular regions over data points which is one of our requirements. However, their goal is not to locate interesting regions with respect to a given statistic but to group data points together for efficient access. Hence, the produced regions only consider the spatial distance of data points and produce fixed regions which the user can use. For any subsequent region, that a user wants to identify based on a threshold or a region size requirement, the user only has access to those fixed regions which could be inadequate.

Finding regions has been considered in other domains [41, 84, 132, 17], with different objectives and algorithms which consider smaller datasets $N < 200,000$. SuRF, is used with an arbitrarily large number of data points N as effectively makes no use of the underlying database system; instead, SuRF uses ML models to perform computations over surrogate models.

4.7 Conclusions

In this chapter, we move a step futher in automating exploratory analysis by employing QDL to the task of automatically identifying interesting regions in data. Specifically, we have introduced SuRF, a solution based on multimodal evolutionary optimization and QDL, which efficiently mines regions of interest in multidimensional datasets. To be precise, the regions are associated with a statistic of interest y computed using the data points included in a region. Thus, the problem of locating regions of interest is formulated as finding regions complying with $y > y_R$ or $y < y_R$, where y_R is a user defined threshold. Given this constraint an optimization problem was introduced which yields a multimodal solution space. SuRF leverages GSO built for this class of optimization problems. SuRF uses QDL to approximate functions for predicting statistic y over interesting regions. By using this approach, SuRF locates regions of interest $150\times$ faster than the best competitor and more than 3 orders of magnitude than the worse, with minimal impact in accuracy. To our knowledge, the problem of finding interesting regions by fusing multimodal optimization with ML has not been investigated before. SuRF is a promising approach in solving a laborious and often manual mining task.

Chapter 5

Dynamic Data & Query Workloads Adaptation

5.1 Introduction

In the last chapters, we have described mechanisms that based on QDL, have managed to expedite large parts of the Data Analysis process. With QDL, we have effectively created ML models that can predict the results of aggregate queries. Given an input query, $\mathbf{q}$, a model based on QDL is simply a mapping of $\hat{f} : \mathbb{R}^d \rightarrow \mathbb{R}$ that imitates an AF. An AF is considered as the *true* function f and a key objective throughout this thesis was to have $\hat{f}(\mathbf{q}) \approx f(\mathbf{q}), \forall \mathbf{q}$. The models were trained using a set of query-answer pairs, $\mathcal{C} = \{(\mathbf{q}, y)\}$. However, as time goes on, the state of a database might change. More data might be inserted, some of the data might be deleted and other parts of the data might be updated. This fact could cause the models to become inaccurate for a subset of all possible queries.

Moreover, the diversity of data analysts and the changing business directives make the query workload more dynamic. The model that was trained to represent an AF, might not sufficiently represent queries issued by all data analysts. By the introduction of new analysts to the team, the distribution of $p(\mathbf{q})$ might change. The same effect could be caused by changing interests, that would denote different queries being issued. This fact would cause significant problems to model $\hat{f}$, as the model $\hat{f}$ has learned to represent distribution $p_{\text{old}}(y|\mathbf{q})$. If we have a condition, where $p_{\text{new}}(y|\mathbf{q}) \neq p_{\text{old}}(y|\mathbf{q})$, then the model could become inaccurate.

Hence, we need to introduce mechanisms for adapting existing models (or creating new ones) when faced with a changing state in the database. Specifically, this chapter can be summarized by the following points:

1. We introduce concepts such as data shift and workload shift. The former indicates a change within the data and the latter a dynamic workload.

2. A method for adapting to data shift is described.

3. An ensemble method based on query-space partitioning is described, to address, initially, diverse analyst interests.

4. Mechanisms for detecting workload shifts and for adapting models are described.

5. Extensive evaluation of the proposed methods for detecting workload/data shifts is presented.

5.2 Definitions

We revisit some of the definitions that we have introduced in earlier chapters and also introduce some new ones that will help throughout this chapter.

Definition 5.2.1. *(Data)* Data corresponds to a collection of random row vectors $\mathbf{a} = [a_1, \ldots, a_d]^\top \in \mathbb{R}^d$. Data are generated from an unknown distribution $p(\mathbf{a})$.

Definition 5.2.2. *(Query)* A query $\mathbf{q}$, is a tuple $(\mathbf{m}, y)$, where $\mathbf{m} \in \mathbb{R}^d$ is a vector that includes the extracted parameters of a query and $y \in \mathbb{R}$ is its corresponding result. Parameter values are generated from an unknown distribution $p(\mathbf{m})$ and the results from an unknown conditional distribution $p(y|\mathbf{m})$. Please note that the type of vectorization process that generates $\mathbf{m}$ can be any of the described processes in the past chapters.

Definition 5.2.3. *(Model)* Assuming an AF is the *true* function f, then a model is an estimated function $\hat{f}$ that is obtained using QDL. Both functions are essentially mappings $\hat{f} : \mathbf{q} \in \mathbb{R}^d \rightarrow y \in \mathbb{R}$ and generally we want a model which minimizes EPE such that $\hat{f} = \arg\min_{\hat{f}} \mathbb{E}[(f(\mathbf{q}) - \hat{f}(\mathbf{q}))^2]$.

Definition 5.2.4. *(Data Shift)* A data shift occurs when the distribution $p(\mathbf{a})$ changes, such that $p_t(\mathbf{a}) \neq p_{t+1}(\mathbf{a})$, consequently we expect that $p_t(y|\mathbf{m}) \neq p_{t+1}(y|\mathbf{m})$ would also hold.

Definition 5.2.5. *(Workload Shift)* Workload shift corresponds to the case where $p_t(\mathbf{m}) \neq p_{t+1}(\mathbf{m})$. Which we expect would cause $p_t(y|\mathbf{m}) \neq p_{t+1}(y|\mathbf{m})$. Hence, any model that has learned the distribution $p_t(y|\mathbf{m})$ could be obsolete.

5.3 Dynamic Data Adaptation

Over the course of time there might be *significant* data updates that invalidate the patterns learned by the models so far. However, in general, new data might not cause the accuracy of models to deteriorate as it is still able to generalize. Hence, data updates need to be

significant, to cause a *Data Shift*. Although insertions/updates/deletions could be expected to be frequent, $p(y|\mathbf{m})$ might not change. Therefore, the key observation is that we do not need to track changes in the data space but instead need to monitor for changes in $p(y|\mathbf{m})$. To tackle *Data Shift*, we could naively retrain the models at fixed time intervals, to be sure that the most updated data are used. Over, 1M+ queries are executed daily in large deployments [70]; thus, it is easy to find new queries executed on fresh data. However, tracking when this event actually happens would help minimize re-training models.

As repeated throughout this thesis, our solutions are based on QDL and do not access data at any time. Therefore, to detect changes in data, we monitor queries that are successively executed by the data warehouse $(\mathbf{q}_t, \ldots)$. To detect changes to the aggregates distribution $p(y|\mathbf{m})$ we employ the two-sample Kolmogorov-Smirnov (KS) test. The KS test output statistic is listed in (5.1)

$$D = \sup_y |F_1(y) - F_2(y)| \tag{5.1}$$

Where F_1 is the Empirical Cumulative Distribution Function (ECDF) at time t of answers $\mathcal{Y}_{1:t}$[1] of all queries that were used to train a model and F_2 is the ECDF of $\mathcal{Y}_{t+1:}$ from monitored queries executed against the data warehouse. The KS test, evaluates the hypothesis that the two sets of answers come from the same distribution. The hypothesis is rejected at a significance level $\alpha \in (0, 1)$ if $D > c(a)\sqrt{\frac{n+m}{nm}}$, where $c(a) = \sqrt{-\frac{1}{2} + \ln \frac{\alpha}{2}}$ and $n = |\mathcal{Y}_{1:t}|$, $m = |\mathcal{Y}_{t+1:}|$. The minimal bound of this test becomes lower to larger sample sizes m, n. The hypothesis is not rejected for a p-value $> \alpha$. If the hypothesis is rejected, then the distribution has shifted and the model associated with this AF needs to be retrained.

5.4 Query Space Partitioning

Recent research, analyzing analytics workloads from various domains, has shown that queries within analytics workloads share patterns and their results are *similar* having various degrees of overlap [137]. Based on this evidence, we mine existing queries to discover clusters of queries, having similar predicate parameters. This helps address the challenge of numerous analysts issuing queries, with those analysts having diverse interests. Hence, by partitioning the query space we proactively adapt our models to diverse query workloads.

Formally, consider a discrete time domain $t \in \mathbb{T} = \{1, 2, 3, \ldots\}$, where at each time instance t an analyst issues a query $\mathbf{q}_t$. The query is executed and an answer y_t is obtained, forming the pair $(\mathbf{q}_t, y_t)$. The issued queries are stored in a growing set $\mathcal{C}_t = \{(\mathbf{q}_1, y_1), \ldots, (\mathbf{q}_t, y_t)\} = \mathcal{C}_{t-1} \cup \{(\mathbf{q}_t, y_t)\}$. Given this set, we incrementally extract knowledge from the query vectors and then train *local* ML models that predict the associated outputs given new, unseen

[1]The notation $1:t$ denotes the answers of queries at time-steps 1 to t and $t+1$: denotes answers of queries from $t+1$ onwards.

queries. This is achieved by on-line partitioning the vectors $\{\mathbf{q}_1, \ldots, \mathbf{q}_t\} \in \mathcal{C}_t$ into disjoint clusters that represent the query patterns of the analysts (fundamentally, within each cluster the queries are much more *similar* than the queries in other clusters). The distance between queries, quantifies how close the predicate parameters are in the vectorial space. Close queries $\mathbf{q}$ and $\mathbf{q}'$ are grouped together into K[2] clusters with respect to $\|\mathbf{q} - \mathbf{q}'\|_2^2$. The objective is to minimize the expected quantization error $\mathbb{E}[\min_{k=1,\ldots,K}\|\mathbf{q} - \mathbf{w}_k\|_2^2]$ of all queries to their closest cluster *representative* $\mathbf{w}_k$, which reflects the analysts query patterns and best represents each cluster. The K query representatives $\mathcal{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$ optimally quantize $\mathcal{C}_t$ minimizing the expected quantization error while each query $\mathbf{q}$ is projected onto its closest representative $\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathcal{W}}\|\mathbf{q} - \mathbf{w}\|_2^2$. Based on the partitioning of $\mathcal{C}_t$, we produce K query-disjoint sub-sets such that $\mathcal{C}_k \cap \mathcal{C}_l \equiv \emptyset$ for $k \neq l$ and $\mathcal{C}_k = \{(\mathbf{q}, y) \in \mathcal{C}_t | \mathbf{w}_k = \arg\min_{\mathbf{w}\in\mathcal{W}}\|\mathbf{q} - \mathbf{w}\|_2\}$. A *local* ML model is then trained over each subset using the pairs in $\mathcal{C}_k, k \in [K]$.

5.4.1 Ensemble Model Prediction

Each aggregate result y from the pair $(\mathbf{q}, y) \in \mathcal{C}_t$ represents the *true* response. Using QDL we train regression algorithms to minimize the EPE. After having partitioned the query space into clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$, we therein train K local ML models, $\mathcal{M} = \{\hat{f}_1, \ldots, \hat{f}_K\}$ that associate queries $\mathbf{q}$ belonging to cluster $\mathcal{C}_k$ with their outputs y. Each ML model $\hat{f}_k$ is trained from query-response pairs $(\mathbf{q}, y) \in \mathcal{C}_t$ from those queries $\mathbf{q}$ which belong to $\mathcal{C}_k$ such that $\mathbf{w}_k$ is the closest representative to those queries. Given a query $\mathbf{q}$ only the most representative model $\hat{f}_k$ is used for prediction, corresponding to the closest $\mathbf{w}_k$:

$$\hat{y} \;=\; \sum_{k=1}^{K} \mathbf{I}_k \hat{f}_k(\mathbf{q}) \tag{5.2}$$

where $\mathbf{I}_k = 1$ if $\mathbf{w}_k = \arg\min_{\mathbf{w}\in\mathcal{W}}\|\mathbf{q} - \mathbf{w}\|_2^2$; 0 otherwise.

5.5 Query Pattern Change Detection

Suppose that all trained ML models $\{\hat{f}_k\}_{k=1}^K$ are deployed into production, we call this phase, the `PREDICTION` mode. That is, for each incoming query, a model $\hat{f}$ can predict the answer of a query *without* executing the query. If we assumed a stationary query pattern distribution, in which queries and analysts' interests do not change, then no adaptation mechanisms would be necessary. However, this is not realistic, as it is highly likely that *analysts interests* change over time (e.g., during exploratory analytics tasks, which are considered as ad-hoc

[2]The number of clusters K is automatically identified by the clustering algorithm used. [7]

processes [66]). So, dynamic workloads may render the $\{\hat{f}_k\}_{k=1}^{K}$ models obsolete, as they were trained using *past* query patterns following distributions which may now be different. Accommodating such dynamics is becoming increasingly important as ML is widely adopted in software in production [119]. Specifically, when referring to analysts' interests, we refer to analysts who are tasked with informing different business decision processes. If those tasks change, the data subspaces to be analyzed become different, which results in changed query patterns. If models cannot be adaptive, expected prediction errors can become arbitrarily high. When $p_t(y|\mathbf{q})$ changes to $p_{t+1}(y|\mathbf{q})$, it is highly likely that any previous approximation would produce high-error answers, unless $p_t(y|\mathbf{q}) \approx p_{t+1}(y|\mathbf{q})$. We capture such dynamics using *concept drift* detection [131, 39], many methods have been developed for adjusting when this arises [52, 39].

We introduce a Change Detection Mechanism (CDM) and an ADaptation Mechanism (ADM), to address this problem, raising a number of challenges: (1) How to detect a query pattern change; we need to enable triggers that alert the mechanism being in prediction mode in case of a *concept drift*; (2) What kind of action should we take in case that happens, i.e., what strategy to follow for updating the ML models; We explore these challenges and describe the decisions we take in tackling them in the remainder.

5.5.1 Change Detection Mechanism

So far, we have trained K different local ML models to predict answers involving only the k-th model that best represents a new incoming query through the representative $\mathbf{w}_k$. This requires to individually monitor whether the query representatives, used for prediction via their respective models, are *still* representatives in long-term predictions or whether the analysts' query patterns have changed. In this case, we need to introduce a CDM that triggers when the original query representative has significantly diverged from the estimated one.

Our approach can be best understood by first assuming that the CDM maintains an on-line average of the prediction error $(y - \hat{y})^2$ such that : $u_k \approx \mathbb{E}[(y - \hat{y})^2|\mathbf{q}]$. This is done for each query representative $\mathbf{w}_k$ across different users. Should the expected error u_k escalate significantly, then this may signal that a query pattern has shifted around the 'region' represented by the representative $\mathbf{w}_k$. But, recall that during `PREDICTION` mode, the actual y is unknown since our goal is to predict accurate answers but without executing the query itself. Hence, we develop an approximation mechanism for change detection, not requiring query executions.

Once we have trained the individual ML models $\mathcal{M}$ and calculated their expected prediction accuracy (using an independent test sample drawn from the original set of queries) we obtain the EPE, which will be constant across all possible queries associated with a particular query

representative defined as: EPE $= \mathbb{E}\Big[\big(f(\mathbf{q}) - \sum_{\kappa=1}^{K} \mathbf{I}_\kappa \hat{f}_\kappa(\mathbf{q})\big)^2\Big]$. Using the EPE, we wish to find a fine-grained estimate of the true prediction error rather just assuming this is constant for each and every unseen query.

To do this, we have analyzed the error behaviour under changing query patterns. Our findings reveal an interesting fact: The Euclidean distance $d(\mathbf{q}, \mathbf{w}_k) = \|\mathbf{q} - \mathbf{w}_k\|_2^2$ of a random query $\mathbf{q}$ from its closest query representative $\mathbf{w}_k$ is strongly correlated with the associated prediction error $(y - \hat{y})^2$.[3] Considering the correlation between $d(\mathbf{q}, \mathbf{w}_k)$ and the local u_k, we define a distance-based prediction error $\tilde{u}_k$ of a query $\mathbf{q}$ as:

$$\tilde{u}_i = \ln\left(1 + d(\mathbf{q}, \mathbf{w}_k) - \min_{\mathbf{q}_\ell \in \mathcal{C}_k} d(\mathbf{w}_k, \mathbf{q}_\ell)\right) \cdot u_k, \tag{5.3}$$

where the natural-log operator acts as a penalizing/discount factor for queries given their distance from the closest representative $\mathbf{w}_k$. The second term within the natural-log operator, $\min_{\mathbf{q}_\ell \in \mathcal{C}_k} d(\mathbf{w}_k, \mathbf{q}_\ell)$ is the minimum distance between the query representative $\mathbf{w}_k$ and the associated queries $\mathbf{q} \in \mathcal{C}_k$. We subtract the minimum distance from $d(\mathbf{q}, \mathbf{w}_k)$ so that the scale of the numbers will not affect the computation of the error.

We base our novel CDM in (5.3) using the series of error approximations $\{\tilde{u}_t\}$ for monitoring concept drifts in query patterns during prediction mode without executing the queries.

Consider the incoming unseen (random) queries $(\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_t)$ arriving in a sequence $t \in \mathbb{T}$ during prediction mode. They are answered by specific local ML models $(\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_k)$, generating a series of distance-based error estimations $\{\tilde{u}_t\}$, $t \in \mathbb{T}$. The CDM monitors this series and, based on a specific threshold, signals the existence of *concept drift*, i.e., checks whether the probability distribution of the queries has changed. Based on the series of error estimations, we learn two query distributions: (1) the *expected* query distribution, which is represented by the query representatives and (2) the *novel* query distribution, which cannot be represented by the current query representatives. The expected distribution $p_0(\tilde{u})$ is estimated given a training period from $\tilde{u}_k$ values corresponding to queries with closest representative $\mathbf{w}_k$. The novel distribution $p_1(\tilde{u})$ is estimated from $\tilde{u}_m$ values corresponding to error values derived from the *rival* representatives $\mathbf{w}_m$ of queries with closest $\mathbf{w}_m$ and $k \neq m$. Based on this, we estimate the distribution of the error values generated from representatives which were not the closest to the queries, thus, approximating novel error values. Both distributions were approximated by fitting the $p(\tilde{u}) \sim \Gamma(e_1, e_2)$ distribution with scale e_1 and shape e_2.

Given a $\tilde{u}_t$ value, we calculate the likelihood ratio $s_t = \log \frac{p_1(\tilde{u}_t)}{p_0(\tilde{u}_t)}$ and the cumulative sum of s_t up to time t, $U_t = \sum_{\tau=0}^{t} s_\tau$. Based on the sequential ratio monitoring for a progressive concept drift in distribution [55] from p_0 to p_1, a decision function is introduced for signaling

[3] A 0.3 Pearson's Correlations was obtained on a real dataset.

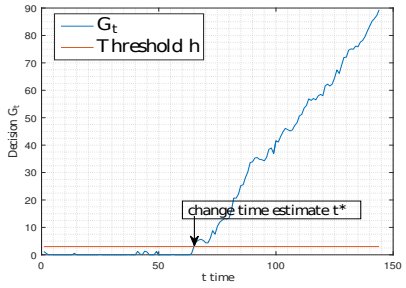

Figure 5.1: Change detection based on the likelihood ratio of the distance-based error, triggered when query patterns are shifted.

a potential concept drift expressed as:

$$G_t \;=\; U_t - \min_{0 \leq \tau \leq t} U_{\tau-1}. \tag{5.4}$$

The decision function in (5.4) indicates the current cumulative sum of ratios minus its current minimum value. This denotes that the *change time* estimate t^* is the time following the current minimum of the cumulative sum, i.e., $t^* = \arg\min_{0 \leq \tau \leq t} U_\tau$. Therefore, given that $U_t = U_{t-1} + s_t$, the decision function in (5.4) is re-written in a recursive form: $G_t = \{G_{t-1} + s_t\}^+$ with $\{z\}^+ = \max(z, 0)$ setting, by convention, $U_{-1} = 0$ and $G_{-1} = 0$. Hence, a concept drift of query patterns projected over the query representatives space is detected at time t_D: $t_D = \min\{t \geq 0 : G_t > h\}$. The parameter h is usually set $3\sigma \leq h \leq 5\sigma$ with σ the standard deviation of $\tilde{u}$. The process is shown in Figure 5.1, the cumulative sum of ratios exceeds the threshold h as soon as queries are issued from an unknown distribution, as the error estimates become steadily larger and are not just random fluctuations in errors. It is worth noting that the change in query distribution is based on fusing the distance between the queries and their closest representatives scaled with the EPE. We refer to this as an indication of degradation in the performance of the model. Given that a change has been detected, the CDM signals the ADM which transits from `PREDICTION` mode to `BUFFERING` mode as shown in Figure 5.2. As soon as a change is detected the CDM signals the ADM component, that new query patterns have been detected. In turn, the ADM signals the *Prediction Component* (containing the $\mathcal{M}$ and $\mathcal{W}$) to be put in `BUFFERING` mode since the prediction component can no longer provide reliable answers for *all* queries. However, we can still leverage the complete system to ask queries following the already known distributions with *only* queries following the new shifted distribution being executed at the data warehouse. By entering `BUFFERING` mode our ADM starts to adjust for the new query patterns until *converging*. At that point it signals the *Prediction Component* to switch back to `PREDICTION` mode, resuming normal operation.

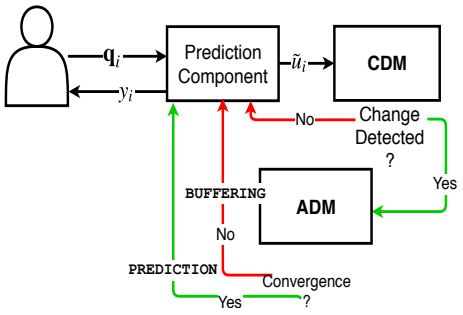

Figure 5.2: Overview of the operation of CDM and ADM which control when estimations can be reliably given to the analyst

5.6 Adaptation Mechanism

In this section, we explain the fundamentals of the ADM. Once a local model $\hat{f}_k$ transits to BUFFERING mode, it is deemed unreliable to accurately predict the answers of incoming queries. Therefore, during this phase, queries should be executed and their actual answers returned to analysts while also being used for adapting the model and representative $(\hat{f}_k, \mathbf{w}_k)$. In the beginning of BUFFERING mode, we introduce a *new* query representative $\mathbf{w}_{K+1}$ and adjust its coordinates with respect to queries that become associated to it. In the end, all queries that have become associated with this new representative are used to train a new model $\hat{f}_{K+1}$.

To reduce the expected number of queries executed during BUFFERING mode, we introduce a *query execution selectivity* mechanism based on the current estimated error in (5.3). Specifically, there would still be some queries issued by an analyst that could be locally answered by current models during that phase. Therefore, we still monitor incoming queries and *discriminate* between two types: (1) the ones that can be locally answered by models in $\mathcal{M}$ and (2) the ones that cannot be answered, since these queries are not well represented by the cached query representatives. The latter queries are then forwarded for execution. The selectivity mechanism relies on the following rule: an incoming query $\mathbf{q}_t$, during BUFFERING mode is locally answered, if the *new* local representative, notated by $\mathbf{w}_{K+1}$, is not the closest representative i.e., $\mathbf{w}_{k^*} = \arg\min_{\mathbf{w} \in \mathcal{W} \cup \{\mathbf{w}_{K+1}\}} \|\mathbf{q}_t - \mathbf{w}\|_2$ and $k^* \neq K+1$. If the query is closest to the non-yet converged novel representative, $\mathbf{w}_{K+1}$, then it is forwarded for execution. However, since the novel representative $\mathbf{w}_{K+1}$ is not converged, we also consider the distance from its rival (second closest) converged representative as a backup. The rival representative can provide assistance and answer the query locally, if it is close enough to include $\mathbf{q}_t$ in the range around its variance σ^2.[4] An example is shown in Figure 5.3, queries $\mathbf{q}_t$ and $\mathbf{q}_{t+1}$ both have $\mathbf{w}_{k+1}$ as the closest representative. However, only $\mathbf{q}_{t+1}$ will be forwarded

[4]This is associated with the vigilance parameter in Adaptive Resonance Theory dealing with the bias-plasticity dilemma.

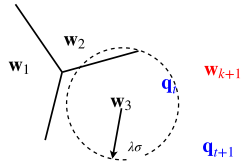

Figure 5.3: Demonstration of the *forwarding rule* by the ADM. Both queries (in blue) have $\mathbf{w}_{k+1}$ (the non-converged) as the closest representative. Only one of them is send to the CS

as $\mathbf{q}_t$ is within the radius of $\mathbf{w}_3$. The forwarding selectivity mechanism is also evident in Algorithm 2. Based on the centroid theorem of convergence in vector quantization, i.e., the converged $\mathbf{w}_k$ is the expected query (centroid) of those queries having $\mathbf{w}_k$ as their closest representative, we exploit the variance $\sigma^2 = \frac{1}{|\mathbb{Q}_k|}\sum_{\mathbf{q}\in\mathbb{Q}_k}\|\mathbf{q}-\mathbf{w}_k\|_2^2$ for activating the forwarding rule. The rule is based on the rival centroid and is fired if $\|\mathbf{q}_t - \mathbf{w}_k\|_2 > \lambda\sigma$ for any scalar $\lambda > 0$ given that the query is closest to the non-converged $\mathbf{w}_{K+1}$. The probability of forwarding incoming queries for execution, given that they cannot be reliably answered by the local model $\hat{f}_k$ is upper bounded as provided in Theorem 3. The query is executed if inevitably the rival representative cannot be used for prediction since $\|\mathbf{q}_t - \mathbf{w}_{K+1}\|_2 > \lambda\sigma$. The value of λ is adopted from the scaling factor of h, i.e., $3 \leq \lambda \leq 5$.

Theorem 3. Given a random query $\mathbf{q}$ whose distance from its rival (second closest) representative $\mathbf{w}_k$ is greater than $\lambda\sigma$, the upper bound of the forwarding probability for query execution is $O(\frac{1}{\lambda^2})$.

Proof. Let the query $\mathbf{q}$ being projected to its closest representative $\mathbf{w}_{K+1}$, which is not yet converged and let its second closest be the converged $\mathbf{w}_k$. The representative $\mathbf{w}_k$ corresponds to the mean vector of those queries belonging in the cluster $\mathcal{C}_k$. In order for the query $\mathbf{q}$ to be forwarded for execution, it means that $\mathbf{w}_k$ should not be the mean vector for the incoming query $\mathbf{q}$. This is indicated if the distance $\|\mathbf{q}-\mathbf{w}_k\|_2$ is greater than a proportion of the norm of the variance σ of the cluster $\mathcal{C}_k$ by a factor $\lambda > 1$. Hence, the query is executed, if this distance is greater than $\lambda\sigma$, which is stochasitcally bounded by the factor $1/\lambda^2$ based on Chebyshev's inequality $P(\|\mathbf{q}-\mathbf{w}_k\|_2 \geq \lambda\sigma) \leq \frac{1}{\lambda^2}$. □

5.6.1 Model Adaptation

When a query is selectively forwarded, the process of model adaptation has as follows: for adapting to new query patterns, we rely on the principle of *explicit partitioning* [131, 39], as a natural extension of our strategy using an ensemble of local ML models. To adjust to new query patterns, we train a new model $\hat{f}_{K+1}$ using executed queries and their answers. This is the optimal strategy for expanding the current $\mathcal{M}$ as other methods might lead to

Algorithm 2 Adaptation Mechanism

Input: $\mathcal{M}$ and $\mathcal{W}$
Set buffer $\mathcal{Q} = \emptyset$
while MODE = BUFFERING **do**
  Prediction Component receives query $\mathbf{q}_t$
  $\mathbf{w}_{k^*} = \arg\min_{\mathbf{w} \in \mathcal{W} \cup \{\mathbf{w}_{K+1}\}} \|\mathbf{q}_t - \mathbf{w}\|_2$ ▷ closest
  $\mathbf{w}_k = \arg\min_{\mathbf{w} \in \mathcal{W} \cup \{\mathbf{w}_{K+1}\} - \{\mathbf{w}_{k^*}\}} \|\mathbf{q}_t - \mathbf{w}\|_2$ ▷ rival
  if $\|\mathbf{q}_t - \mathbf{w}_k\|_2 > \lambda\sigma$ and $\mathbf{w}_{K+1} = \mathbf{w}_{k^*}$ **then**
    Send query $\mathbf{q}_t$ for execution
    $\mathcal{Q} = \mathcal{Q} \cup \{(\mathbf{q}_t, y_t)\}$ ▷ actual query-answer pair
    Adapt prototype $\mathbf{w}_{K+1}$
  else
    $\hat{y} = \hat{g}(\mathbf{q}_t)$ ▷ prediction
  end if
  Update learning rate γ
  if convergence w.r.t. c **then**
    Train new model $\hat{f}_{K+1}$ using $\mathcal{Q}$
    $\mathcal{M} = \mathcal{M} \cup \{\hat{f}_{K+1}\}, \mathcal{W} = \mathcal{W} \cup \{\mathbf{w}_{K+1}\}$
    Set MODE = PREDICTION
  end if
end while

catastrophic forgetting [52]. Indicatively, such methods adopt strategies to adapt the current model by adjusting to new patterns whilst forgetting the old ones. In our context, this is not applicable since analysts have the flexibility to issue queries either conforming to the *old* patterns or to the new ones, depending on the analytics process.

The adaptation process is performed with parameters: the K query prototypes $\mathcal{W}$ and their associated ML models $\mathcal{M}$ as shown in Algorithm 2. Let the queries series $\{\mathbf{q}_1, \mathbf{q}_2, \ldots\}$ forwarded, based on selective forwarding. This means that most likely a query $\mathbf{q}_t$ conforms to new query patterns thus sent for execution. Once query $\mathbf{q}_t$ is executed and its actual answer y_t is obtained, it is then considered as a new (initial) representative $\mathbf{w}_{K+1}$ for $\mathcal{M}_{K+1}$. The pairs $(\mathbf{q}_t, y_t)$ are then used to incrementally update $\mathbf{w}_{K+1}$ and buffered in $\mathcal{Q}$, which will be the training set for $\hat{f}_{K+1}$. The adaptation of $\mathbf{w}_{K+1}$ to follow the new query pattern is achieved by SGD [30]. SGD is widely used in statistical learning for training, by considering one example (query-answer) at a time. We focus on the convergence of the query distribution by moving the new query representative towards the estimated median of the queries in $\mathcal{Q}$ and not the corresponding centroid. This is introduced so that the new representative converges to a robust statistic, free of outliers and more reliable than the centroid (mean vector). The convergence to the median denotes with high reliability *convergence to the distribution*, which is what we desire for model convergence. In this case, we provide the adaptation rule of the new query representative to converge to the median of the forwarded queries, in Theorem 4.

Theorem 4. The novel representative $\mathbf{w}_{K+1}$ converges to the median vector of executed queries with respect to the update rule $\Delta\mathbf{w}_{K+1} \propto \gamma sgn(\mathbf{q} - \mathbf{w}_{K+1})$, $\gamma \in (0, 1)$; $sgn(\cdot)$ is the signum function.

Proof. Each dimension i of the median vector $\mathbf{m}$ of the queries $\mathbf{q}$ in a sub-space satisfies: $P(q_i \geq m_i) = P(q_i \leq m_i) = \frac{1}{2}$. Suppose that the new representative $\mathbf{w}_{K+1}$ has reached equilibrium, i.e., $\Delta\mathbf{w}_{K+1} = \mathbf{0}$ holds with probability 1. By taking the expectations of both sides of the update rule $\mathbb{E}[\Delta\mathbf{w}_{K+1}] = \alpha\mathbb{E}[sgn(\mathbf{q} - \mathbf{w}_{K+1})] = \mathbf{0}$ and focusing on each dimension i, we obtain that: $\int sgn(q_i - w_{K+1,i})p(q_i)dq_i = P(q_i \geq w_{K+1,i}) \int p(q_i)dq_i - P(q_i < w_{K+1,i}) \int p(q_i)dq_i = 2P(q_i \geq w_{K+1,i}) - 1$. Since $\mathbb{E}[\Delta w_{K+1,i}] = 0$ is constant, then $P(q_i \geq w_{K+1,i}) = \frac{1}{2}$, which denotes that $w_{K+1,i}$ converges to the median of q_i, $\forall i$. □

Once the novel representative, $\mathbf{w}_{K+1}$, has converged, given the condition $\Delta\mathbf{w}_{K+1} < c$ and a threshold parameter c, then a new model $\hat{f}_{K+1}$ is trained using queries buffered in set $\mathcal{Q}$. The models set $\mathcal{M}$ and representatives set $\mathcal{W}$ are expanded to account for the newly trained model $\hat{f}_{K+1}$ and representative $\mathbf{w}_{K+1}$.

5.6.2 Convergence to an Offline Mode

When the system transits from the `BUFFERING` to `PREDICTION` mode, the enhancement of $\mathcal{M}$ and $\mathcal{W}$ gradually decreases the probability to enter the `BUFFERING` mode in the future. This indicates that the gradually expanding sets reflect the analysts' way of exploring and analyzing data. Because of this expansion, the transition probability from `PREDICTION` to `BUFFERING` mode gradually decreases saving computational resources.

The entry probability β to `BUFFERING` mode decreases as $K \to \infty$ thus reducing execution overhead by transiting to 'offline' mode.

This statement is better understood with an example, shown in Figure 5.4. Imagine a fixed a number of query-space regions Z. That is, at any time, a query could be issued to any of those regions. Assume that queries only have two query parameters, for clarity, which we abbreviate as the i^{th} and j^{th} query parameters. Initially, none of those regions are known and hence $K = 0$ then the probability of entering buffering mode β is equal to the probability of issuing a query to any one of those unknown query subspaces, such that $\beta = 1 - \frac{K}{Z}$. Gradually and as $K \to Z$, this probability diminishes. In Figure 5.4, these query subspaces are gradually uncovered, starting from the top-left and moving to the right. One by one, the query subspaces are identified such that whenever a query is issued with these subspaces, it can be answered by a model $\hat{f}$. By the end of this process, seen in the bottom-right figure, any query that is issued can be adequately represented.

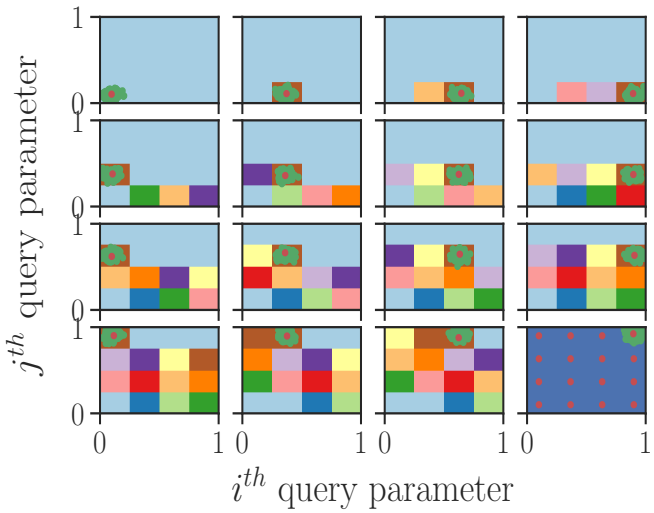

Figure 5.4: Gradually converging to an offline mode. Starting from the top-left and moving to the right, the query-space is uncovered until all queries are represented.

5.7 Evaluation Results

The main questions we are striving to answer in our evaluation are the following :

1. How effective are the CDM/ADM mechanisms and what is the effect of continuously learning and adapting to new queries ?
2. Can we successfully adapt to data shifts ? What is the effect of updates to $p(y|\mathbf{q})$ on error ?
3. How sensitive is the CDM algorithm ?

5.7.1 Implementation & Experimental Environment

To implement our algorithms we used `scikit-learn` , `XGBoost`[35] and an implementation of the `Growing-Networks` algorithm [89]. We performed our experiments on a desktop machine with a Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and 16GB RAM. For the real datasets, the GrowingNetworks algorithm was used for clustering mainly because of its invariance to selecting a pre-defined number of clusters and it's ability to naturally grow (as required by our adaptability mechanisms).

Real datasets: We use the **Crimes** dataset from [6]. The Crimes dataset contains $|\mathcal{R}_1| = 6.6 \cdot 10^6$ data vectors. For Crimes, we generated predicates restricting the spatial dimension as essentially this is what analysts would be doing in exploration tasks. For the predicates

in the spatial dimension we used multiple multivariate-normal distributions to simulate the existence of multiple users.

5.7.2 Query Workload Adaptivity

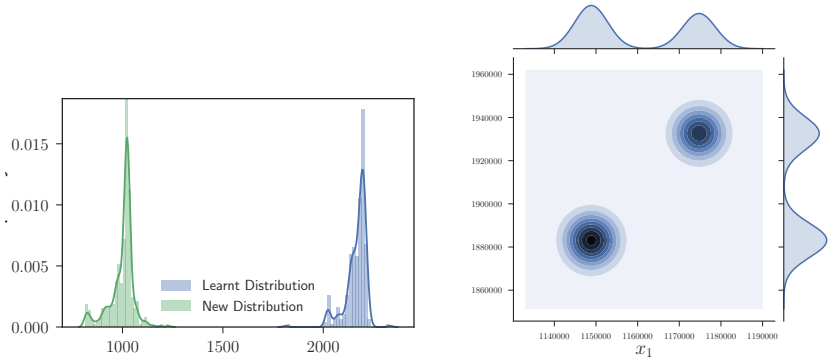

Figure 5.5: (Left) Inducing concept drift of learned distribution of y; (right) Lower left is the initial query pattern distribution; upper right is the new one.

To examine CDM, ADM due to concept drift we devised the following experiment. Consider a $\mathcal{M}_i$ that has already learned a particular distribution of y, being deployed to answer queries. At a particular point in time query patterns might shift as shown in Figure 5.5. Figure 5.5 shows two different query distributions. Figure 5.5(left) are the distributions of the query answers y. Their respective query patterns are shown in Figure 5.5 (right) [5]. Our aim is to examine whether CDM detects a query pattern shift from one distribution to another. If remained undetected, it will cause detrimental problems in accuracy due to different distributions of y. We first set the detection threshold $h = 3\sigma_{\tilde{u}}$ and convergence threshold $c = 0.008$; a following sensitivity analysis shows the impact of these tuning parameters on ADM and CDM. We gradually introduce new query patterns and compare our system with an approach where no adaptation is deployed. Figure 5.6 shows the different queries being processed by our mechanism and the associated *true* prediction error. We first measure the error of queries using the *known* distribution until $t = 66$. From that point onwards, we shift to the unknown distribution and evidently the error increases dramatically should no adaptation mechanism be employed. On the other hand, CDM detects that a shift has happened and transits the system from prediction mode to buffering mode until the exiting criteria are met. In the end, a new model is introduced which is trained using the new distribution as evidenced by the decreased error at $\mathcal{M}_{new}$.

[5]Only two dimensions shown for visualization purposes

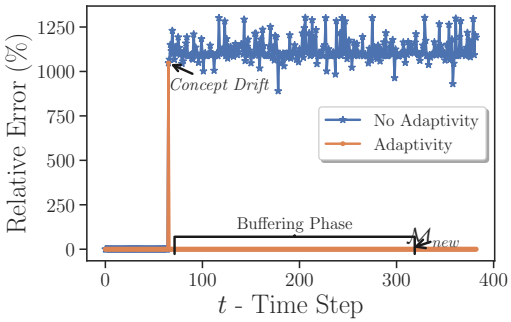

Figure 5.6: Error with concept drift detection/adaptation.

5.7.3 Parameter Sensitivity

Parameters h and c are responsible for the ADM and CDM with the impact of c shown in Figure 5.7(left). As we increase c we allow for an early exit from buffering mode. An early exit, means that less queries have been processed, thus, potentially the examples are not sufficient for accurately learning the distribution. This is witnessed, in Figure 5.7(left), where the relative error increases, therefore the accuracy decreases as we increase c.

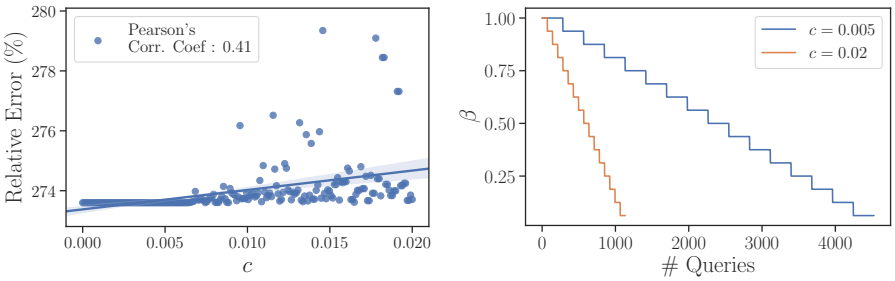

Figure 5.7: (Left) Error vs. convergence c; (right) diminishing probability of buffering β as more query spaces are *known*.

Figure 5.7(right) shows the diminishing probability of entering the buffering mode $\beta = P(G > h)$ building upon our discussion of slowly converging the system into an Offline mode. As more queries are processed across varying query spaces, our system is incrementally learning the whole query space. At a certain point, all query subspaces will be *known* along with their representatives. Thus, the probability of entering the buffering mode due to potentially unknown query distribution reduces to zero almost surely. We provide an experiment in which there is a predefined fixed number of Query Spaces (QS) $Z = 16$, $QS = \{QS_1, \ldots, QS_Z\}$. Queries are generated randomly in a sequence from one QS to another, each time learning QS_{k-1}. Thus, given this fixed number of QSs and setting h, c, the probability of entering buffering mode can be approximated by $\beta = P(G > h) = 1 - \frac{K}{Z}$,

where $K \leq Z$ denotes the number of known QS so far. Liaising this with Figure 5.7, we observe that the probability reduces in a step-wise manner tending to zero when $K \rightarrow Z$. For a relatively high c value, the rate of convergence to the offline mode becomes faster but with a higher error as witnessed by the previous experiment. As for parameter h, a low value indicates smaller tolerance when estimating errors and vice versa. This might force the system to adapt when not needed. Thus, it is domain appropriate to hyper-tune the parameter accordingly. We have found the proposed heuristic of $3\sigma \leq h \leq 5\sigma$ to work well empirically.

5.7.4 Adaptation to Data Updates

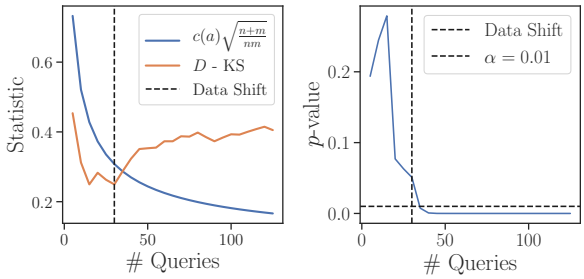

Figure 5.8: (Left) x-axis; Number of queries processed initially from p_1 and then from p_2; y-axis; measured statistic (Right) p-value showing the significance of the KS test.

We also conduct an experiment to assess data shift detection. As we have elaborated we need to identify cases where $p(y|\mathbf{m})$ change by observing queries executed at the data warehouse without accessing any data. Monitoring actual insertions/updates/deletions could prove futile as the distribution of y might not be changing. For this experiment we use two distributions $p_1(\mathbf{m}, y)$ and $p_2(\mathbf{m}, y)$. The distributions for parameters $\mathbf{m}$ are multivariate Normal distributions initialized randomly at the data space of `Crimes` data set and the answers y are the actual answers for `COUNT` over the same data set. We deliberately choose `COUNT` as it is an AF that will most likely change, under significant insertions/updates/deletions. Statistics like `AVG` are more resilient to such changes.

In this experiment we obtain the empirical distribution function of $p_1(y|\mathbf{m})$ and conduct the KS test at regular intervals. We set the significance level at $\alpha = 0.01$ and hence $c(\alpha) \approx= 1.628$. At a specific point in time we shift to $p_2(y|\mathbf{m})$ which we then expect that KS statistic will go over the threshold. The results in Figure 5.8(left) show that as the distribution shifts, the KS statistic (orange line), increases and becomes larger than $c(a)\sqrt{\frac{n+m}{nm}}$, as soon as data shift happens (vertical dotted line). This is also indicated by the p-value shown in Figure 5.8(right). A vertical line is drawn when data shift occurs and a horizontal line is drawn at the specified significance level set $\alpha = 0.01$. We see that the p-value becomes less than α indicating that the statistic is statistically significant.

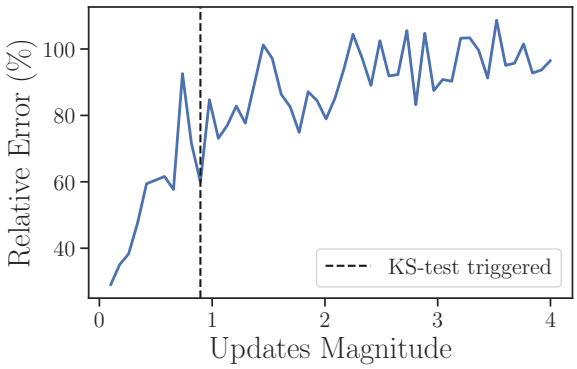

Figure 5.9: As updates become more significant, relative error increases. However, we are able to detect and adapt in such an event.

In addition, Figure 5.9 shows the point where the KS statistic fires, with respect to the impact of updates on $p(y|\mathbf{m})$. Specifically, for this experiment we add a noise component $\mathcal{N}(0, \sigma)$ with $\sigma \in (0.01, 4)$ to the distribution $p(y|\mathbf{m})$, such that $y = p(y|\mathbf{y}) + \mathcal{N}(0, \sigma)$. This simulates larger updates that distort the initial distribution even more. We can see that at the point where the hypothesis is reject (ie data shift occurs), signaled by the vertical line in Figure 5.9, we can proactively adapt our models before the error drastically increases.

5.7.5 CDM Sensitivity Analysis & Robustness

We also conducted experiments to analyse the sensitivity of the CDM to various properties focusing on (a) noisy observations, (b) outliers that might be encountered as part of the original distribution and do not indicate a distribution shift, and (c) skewed distributions in the predictor variables. By examining these properties, one can be advised on the robustness of the CDM.

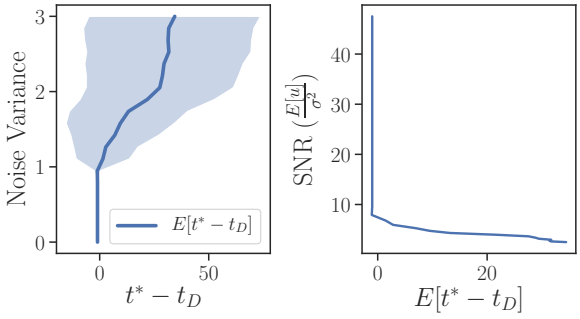

Figure 5.10: (Left) Expected time-point detection difference (x-axis) as the noise intensifies (y-axis). (Right) The expected time-point detection difference with respect to the Signal-to-Noise-Ratio.

We initially conducted an experiment to quantify the impact of noise on the CDM performance. Queries, and hence their estimated errors are sampled from an initial distribution p_1, where $p_1(\tilde{u})$ is the distribution of the estimated errors that are calculated using (5.3). We then introduce white noise into these estimations such as $\tilde{u} \leftarrow \tilde{u} + \mathcal{N}(0, v^2)$, where $\mathcal{N}(0, v^2)$ is Gaussian distribution with zero mean value and finite variance v^2 for $v \in [0, 3]$. We repeat this experiment for 200 random time-points t^*, where a time-point t^* is the time instance at which we switch distribution $p_1(\tilde{u})$ to $p_2(\tilde{u})$, which is expected to trigger the CDM. We then record the exact time-point t_D where the CDM has actually fired and calculate their expected detection difference of $t^* - t_D$, notated as $\mathbb{E}[t^* - t_D]$. Notice that we do not calculate their absolute value difference, as a negative value indicates that the CDM has fired after t^*, while a positive value before t^*. As before, we set the threshold for $h = 3\sigma$, where σ is the standard deviation of $p_1(\tilde{u})$. The results for this experiment are shown in Figure 5.10. Figure 5.10(left) shows the expected time-point difference, $\mathbb{E}[t^* - t_D]$ and the standard deviation across many runs. It is evident that as the noise variance is within $[0, 1]$ the expected time-point detection difference is 0 which indicates that the CDM is robust to noise. However, as the noise intensifies, it effectively shifts the distribution for the estimations $\tilde{u}$ and the CDM fires prematurely on average. This is due to the fact that the noisy observations are now deemed as large error estimations. Nonetheless, this is not to say that the CDM is no longer robust, as we expect this kind of behavior. This is reinforced by measuring the Signal-to-Noise-Ratio (SNR), which we plot in Figure 5.10(right). The SNR is a quantity that measures how much stronger the signal is with respect to the noise. As it is evidenced in Figure 5.10(right), the expected time-point detection difference only increases as the SNR decreases, which indicates that the noise becomes really large and then weakens the signal.

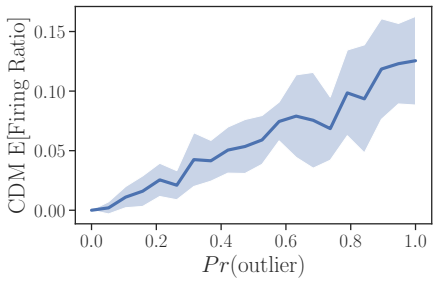

Figure 5.11: The x-axis shows the probability of injecting an outlier to the estimated errors, while the y-axis shows the expected firing ratio for the CDM (lower is better).

We have also investigated the effects of outliers on CDM. We sample values from an initial distribution $p_1(\tilde{u})$ and record whether the CDM has fired. However, we introduce outliers to the sampled values by modifying the estimated error using $\tilde{u} = \tilde{u} + m\sigma$, where σ is the standard deviation of the distribution for $\tilde{u}$ and multiplicative factor $m \in \{3, 4\}$. The value of m is selected based on a Bernoulli trial with success probability 0.2 and a success sets

the value of $m = 4$. An outlier is *injected* based on a Binomial distribution with probability of success p over a number of trials $n = 100$, where $p \in [0, 1]$. We vary the probability p and record the expected firing ratio for a number of runs. We plot the results of this experiment in Figure 5.11. As one can observe, the expected firing ratio is relatively low even for outliers probability $Pr\{\text{outlier}\} = 0.5$, while the expected firing ratio is less than 15% where the $Pr\{\text{outlier}\} = 1$. We can also observe a linear increasing trend on the expected firing ratio with respect to the probability of injecting an outlier. Please note, that even at $Pr\{\text{outlier}\} = 1$, the CDM should not fire as the outlier values for $\tilde{u}$ do not indicate a distribution shift, but *extreme* values of the same distribution. Hence, given this experiment it is safe to conclude on the fact that CDM is robust to outliers.

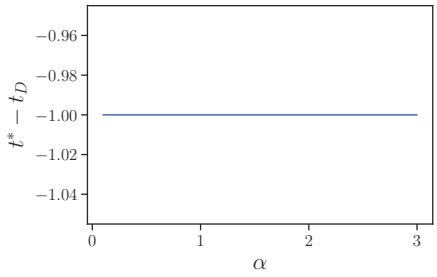

Figure 5.12: The parameter α controls the skewness of the $\Gamma(\alpha, \beta)$ distribution; the skew $= \frac{2}{\sqrt{\alpha}}$ has no visible effect on theCDM.

Lastly, we evaluate the CDM robustness and performance for various highly skewed distributions of the predictors. In this experiment, our aim is to examine whether skewed distributions for the predictors would affect the CDM's accuracy. As the CDM largely relies on a distance-based metric (refer to (5.3)), i.e., CDM is highly influenced by the predictors, a skewed distribution might affect the estimations produced and might cause the CDM to misfire. For this experiment, we use $\Gamma(\alpha, \beta)$ distributions for the predictors. Each predictor is distributed with respect to $\mathbf{q} \sim \Gamma(\alpha, \frac{\beta}{c})$, where $\alpha \in [0.1, 3]$ and parameters β and c are pre-defined values that define the *scale* and *location* of the distribution. Therefore, we initially generate a distribution $p_1(\mathbf{q})$ where $\mathbf{q} \sim \Gamma(\alpha, \frac{\beta_1}{c_1})$. We set the EPE ($u_k$) to a constant and obtain an initial distribution of estimated errors $p_1(\tilde{u})$ based on (5.3). We also generate a second distribution of predictors $p_2(\mathbf{q})$ where $\mathbf{q} \sim \Gamma(\alpha, \frac{\beta_2}{c_2})$ and obtain the distribution of errors $p_2(\tilde{u})$ that we consider as the errors when the distribution of queries has shifted. We then perform the same experiment as before, initially the error estimates from p_1 are passed to the CDM. Then at a random point in time, we switch to p_2 and observe the detection time-point difference $t^* - t_D$. We plot the results of this experiment in Figure 5.12. The α parameter controls the skewness of the predictors distributions. The skewness decreases at a rate of $\frac{2}{\sqrt{a}}$. As it is evidenced, the CDM is unaffected by this as it correctly fires immediately after the time-point t^* where the shift happens.

Overall, through these set of experiments, we have examined the CDM's sensitivity on a number of different settings. We have demonstrated its applicability and robustness under various conditions and concluded on the fact that CDM is appropriate to use in many scenarios.

5.8 Related Work

Query-driven models are largely being deployed for both aggregate estimation [21, 19] and for hyper-tuning [133] database systems. In the past chapters we have also witnessed how QDL can be employed for estimation, exploration and exploitation. In this chapter, we address the crucial problem of detecting data and workload shifts and adapting to them. Which (to our knowledge) has not been addressed before. Hence, our framework can be leveraged by all query-driven implementations in cases of dynamic workloads and changing data distributions that are non-stationary.

Moreover, concept drift adaptation is well understood [131, 52, 39, 44], mostly dealing with classification tasks, where classifiers adapt to new classes. We adapt concept drift to query-driven analytical processing, relying on *explicit partitioning* [52], ensuring it avoids destructive forgetting given that the accuracy for the previously learned query patterns will not degrade. It is also favorable given our initial off-line design which already uses partitioning for clustering the query patterns and learning local models in given sub-spaces. Our work contributes with monitoring and detecting real-time query patterns change based on *approximating* the prediction error, which differentiates with the previous concept drift methods by measuring the actual error; evidently, this is not applicable in our case.

5.9 Conclusions

In this work we contribute a novel framework for adapting trained models under workload-/data shifts. We focus on models used for estimating analytical query answers efficiently and accurately, however we note that the framework is applicable in other domains as well. The contributions are centered, on a novel suit of ML models, which mine past and new queries and incrementally build models over quantized query-spaces using a vectorial representation. The described adaptation mechanisms bear the ability to adapt under changing analytical workloads and data distributions, while maintaining high accuracy of estimations. As shown by our evaluation, the proposed approaches are highly accurate and robust. The contributed CDM and ADM mechanisms are able to detect changes using estimated errors and swiftly adapt models. Furthermore, as more queries are processed, our system has the potential to reach global convergence as no more query patterns remain undiscovered. This

can significantly reduce unnecessary communication to cloud providers thus reduce network load and monetary costs.

Chapter 6

Conclusion

6.1 Overview

In this thesis, we have examined how QDL can be applied to expedite and automate sub-processes during Data Exploration. To do that, we have drawn inspiration and made use of a lot of techniques covered in the fields of Data Management, Machine Learning and Data Mining. Specifically, QDL has been applied in the contexts of query-answer *estimation*, AQ explanation to enhance *exploration* and finally for automating the process of identifying regions of interest. We have covered a lot of complementary topics to be able to apply QDL in varying contexts. Some of these topics include but are not limited to : Supervised Regression [46], AQP [12], QR[76], Explanations of AQs [113], Unsupervised Clustering [46], MARS [48], Locally Piecewise Linear models [46], Optimization, Interesting Region Identification [24], Evolutionary Optimization/Multi-modal Optimization [79], Concept Drift Detection and Adaptation [131].

The contributions of this thesis are crucial as data are exponentially increasing making the data exploration process long. In addition, generating insights becomes more expensive both in terms of computational power and monetary costs. As data are often stored in the Cloud, data analysis is costly. This is due to the fact that, the nature of data exploration, requires data analysts to repetitively execute different queries, with each query carrying a cost. Moreover, data analysts do not have tools that can assist them during data exploration, therefore it becomes an exhaustive process where queries are issued and refined in an iterative manner. QDL, and our contributions offer alternative methods that could be employed, offering some assistance to the data analysts. Finally, most of the current approaches (in any of the contributed domains) are data-driven. This means that AQ *estimations*, AQ *explanations* and interesting region identification are tasks that are performed using data. Inevitably, any of these approaches does not scale as data increases. With QDL, we have offered an alternative route and have shown how each one of these tasks can be performed *accurately* and

efficiently.

Significant research contributions were made, as we have shown a new way which decouples data exploration tasks from data and instead leverages knowledge obtained from queries. As this thesis described different contexts in which QDL was applied successfully, we remain confident that its application will soon follow in other contexts. In addition, we have addressed one of the crucial challenges of deploying QDL based systems, namely dynamic data and changing query workloads. Hence, future practitioners or researchers wishing to use QDL can utilize our contributed techniques for adapting their systems in such cases.

In general, QDL can be applied in domains where efficiency is paramount. Data-driven algorithms do not scale as data rapidly increase and algorithms (such as the ones described in the context of interesting region identification) are extremely expensive to execute over vast amounts of data. Hence, QDL offers great scalability when data size makes these algorithms unsuitable.

6.2 Lessons Learned & Future Work

We have had to face and overcome a number of challenges while developing QDL and applying it to various domains. This section will be a summary of all the things that we have learned throughout this journey. In addition, we describe future goals that will be useful to attain, as QDL becomes more widespread.

6.2.1 Finding the right evaluation methods

Firstly, it is very important to note that throughout this thesis we have described QDL being applied in different contexts. Although, the underlying framework is the same in all of these contexts, the evaluation procedure, metrics and criteria are vastly different. For instance, in *Estimation*, it is crucial, to obtain highly accurate answers as efficiently as possible. The accuracy in this case was measured by *Relative Error* and the chosen model performed best in terms of this error. However, in both *Exploration* and *Exploitation* chapters, accuracy is measured by different criteria as what we strive to accomplish is different. Therefore, future practitioners should take care to properly define metrics/criteria/evaluation procedures that are suitable to the task hand and not resort to only measuring the point-wise distance of $|y - \hat{y}|$.

6.2.2 Identifying proper ML models

We have made extensive use of GBMs in both *Estimation* and *Exploitation*. Such models, offer great scalability, accuracy and efficiency. However, we initially examined other ML models. Specifically, we investigated the use of Deep Learning for the tasks at hand. Most often, Deep Learning models offered comparable accuracy but the overhead of properly tuning and training them made their use inappropriate. In addition, in some cases GBMs were also inappropriate. For instance, in *Exploration*, we make use of linear models as they are highly interpretable and more appropriate to our use case. Finally, we had initially focused on coming up with new ML models that were finely-tuned for each use case. Inevitably, this caused lots of overhead and proved to be a distraction from our main focus. Hence, we chose not to reinvent the wheel and instead use models that have been empirically shown to work well. However, in many cases, we had to employ our own techniques to make them suitable. For instance, in *Estimation*, coming up with a vectorization process for SQL queries and identifying ways to obtain error estimates was not trivial. In addition, in *Exploration* we had to incorporate, relatively simple models, in a complex strategy that used an ensemble of such simple models.

6.2.3 Lack of Query Workloads

As real query-workloads are often not available we resorted in creating our own synthetic workloads and shared them with the community [9]. Our approach was to generate varying workloads with realistic conditions, simulating the existence of multiple analysts with varying interests. However, in the future, real query workloads will be needed for a more truthful overview of how analysts interact with data and how often workloads/data change. But it is our understanding, that this might be hard to achieve, as organizations are relatively cautious when sharing such sensitive information. In addition, current privacy regulations might impose additional barriers to publicly sharing this type of information.

6.2.4 Alternative Vectorization Process

Throughout all chapters, we have used various vectorization processes. As a reminder, a vectorization process, is the process by which we transform a query to a vector. For instance, the vectorization process in *Estimation* is different than the one described in *Exploration* and *Exploitation*, as the focus in each chapter is different. However, in all cases, this was a manual process. We had to design this transformation procedure and make sure it worked correctly. Recent advances in Natural Language Processing [93] and Deep Learning [53] might be promising approaches in automating this procedure.

6.2.5 Active Learning

As previously discussed, real query-workloads are lacking and our current approach was to generate our own synthetic datasets. This is a manual process, with many (realistically valid) assumptions. However, as different datasets are guaranteed to have varying multivariate distributions, a single workload generation strategy is difficult to generalize over multiple datasets. Hence, a drastically different approach, which automates much of this process, is the way forward. Active Learning [121], is a technique in which the agent/model gets to choose between training examples and can also identify *regions* that should be queried next. This is a fairly promising alternative, to the manual process that is currently being employed. It can also prove to be more robust to changing datasets, as the model gets to choose which regions to query. Inherently, this would allow to issue more queries in areas where AQ results vary more and less queries in areas where AQ results have less variation.

6.2.6 System Implementation

We have verified the utility of all of the proposed methods individually, however, implementing all of the techniques under one umbrella system will be extremely beneficial. A single system will give the opportunity to holistically expand the analytics capabilities of data analysts. For instance, analysts could leverage ML-AQP to increase efficiency whilst executing aggregate queries and at the same time consult explanation functions as to what kind of query to execute next. Another user could leverage SuRF to identify interesting regions and then utilize ML-AQP for further exploration within that region. Having all of this centrally available by a single system and making all of the individual techniques interoperable could dramatically improve data analysts experience during data exploration.

6.3 Concluding Remarks

Overall, the journey towards automating data exploration does not end here. QDL presents a good alternative to data-driven algorithms for data exploration, and there is still lots of work to be done within this domain. Throughout this thesis, we have examined contexts where QDL can be used effectively along with addressing crucial challenges. The efficiency achieved by using QDL is superior to data-driven approaches in multiple tasks. Therefore, practitioners are urged to incorporate this technique in cases where efficiency is paramount. Researchers wishing to engage with this topic, can consult the sections presented above for some potential challenges within the domain of QDL. Finally, this thesis is a comprehensive guide to QDL for both researchers and practitioners and our hope is to inspire more people working in this new exciting topic.

Acronyms

ADM ADaptation Mechanism. 111, 113–115, 118–120, 125, 136, 137

AF Aggregate Function. 13, 16, 18, 20–27, 29, 31, 33, 38, 40, 43, 107–109, 121, 133, 134

AQ Aggregate Query. 13, 14, 46–58, 62, 63, 66, 67, 70, 80, 83, 127, 130, 133, 134

AQP Approximate Query Processing. 17, 18, 21, 23, 25, 26, 29, 30, 33, 35, 37, 38, 40–42, 44–46, 127

CDF Cumulative Distribution Function. 39, 76, 86, 97, 134

CDM Change Detection Mechanism. 5, 111–114, 118–120, 122–125, 136, 137

CoV Coefficient of Variation. 96

DBMS Database Management System. 7, 13, 48, 62, 63, 66, 80, 83

ECDF Empirical Cumulative Distribution Function. 109

EDA Exploratory Data Analysis. 81

EEL Expected Explanation Loss. 55, 56, 60–62, 64, 71

EPE Expected Prediction Error. 11, 28, 30, 31, 89, 90, 108, 110–113, 124

GBM Gradient Boosting Machines. 29, 41, 129

GSO Glowworm Swarm Optimization. 4, 86–89, 91, 94, 97–102, 106, 136

GT Ground Truth. 92–96, 102, 105, 136

IoU Intersection over Union. 92–96, 100–104, 136, 138

KDE Kernel Density Estimation. 88

KL Kullback-Leibler divergence. 71, 73, 135

KS Kolmogorov-Smirnov. 109, 121, 122, 137

L1-EQE L1 Expected Quantization Error. 59, 62

L2-EQE L2 Expected Quantization Error. 60, 62

LPM Local PLR Model. 3, 56–64, 66–69, 72–77, 134, 135

LR Location Representative. 57–61, 63–66, 69, 134, 135

MAE Median Absolute Error. 40

MARS Multivariate Adaptive Regression Splines. 61, 70, 77

ML Machine Learning. 5–7, 10–13, 16–19, 24–26, 29–31, 33, 40, 41, 44, 45, 50, 51, 53, 55, 66, 70, 77, 82, 83, 88–91, 103–107, 109–112, 115, 116, 125, 129, 136

NRMSE Normalized Root Mean Squared Error. 72, 74, 135

OLAP Online Analytical Processing. 23, 105

PLR Piecewise-Linear Regression. 54–56, 60–62

PSO Particle Swarm Optimization. 86, 87

QDL Query-Driven Learning. 10, 12–14, 16, 38, 44–48, 51, 80, 82–84, 89–91, 105–110, 125, 127, 128, 130

QR Quantile Regression. 31, 32, 41, 44, 127

RMSE Root Mean Squared Error. 102–104, 136

RR Region Representative. 57–61, 63–66, 69, 135

SC Subspace Clustering. 104

SD Subgroup Discovery. 104

SGD Stochastic Gradient Descent. 29, 64, 65, 116

SPA Selection-Projection-Aggregate. 20, 21

SPJA Selection-Projection-Join-Aggregate. 21

SSQE Sum of Squared Quantization Errors. 59, 60, 76, 77

List of Figures

List of Tables

Bibliography

[1] Crimes workload. URL :https://archive.ics.uci.edu/ml/datasets/Query+Analytics +Workloads+Dataset. Accessed: 2019-06-28.

[2] Instacart. URL :https://www.instacart.com/datasets/grocery-shopping-2017. Accessed: 2019-06-28.

[3] Instacart queries. https://github.com/verdictdb/verdict/wiki/Instacart-Queries.

[4] Prim - implementation. URL: https://github.com/Project-Platypus/PRIM.

[5] Tpc-h. http://www.tpc.org/tpch/.

[6] Crimes - 2001 to present. URL: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2, 2018. Accessed: 2018-08-10.

[7] Growing networks. URL: https://github.com/Skeftical/GrowingNetworks, 2018. Accessed: 2018-08-10.

[8] Intel lab data. URL: http://db.csail.mit.edu/labdata/labdata.html, 2019. Accessed: 2019-04-10.

[9] Query analytics workloads dataset data set.
URL: https://archive.ics.uci.edu/ml/datasets/Query+Analytics+Workloads+Dataset, 2019. Accessed: 2019-07-29.

[10] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[11] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 481–492. ACM, 2014.

[12] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[13] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[14] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Gowdal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, et al. Cloudy with high chance of dbms: A 10-year prediction for enterprise-grade ml. *arXiv preprint arXiv:1909.00084*, 2019.

[15] P. Agrawal, R. Arya, A. Bindal, S. Bhatia, A. Gagneja, J. Godlewski, Y. Low, T. Muss, M. M. Paliwal, S. Raman, et al. Data platform for machine learning. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1803–1816, 2019.

[16] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 153–164. ACM, 2011.

[17] C. Anagnostopoulos and S. Hadjiefthymiades. Intelligent trajectory classification for improved movement prediction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(10):1301–1314, Oct 2014.

[18] C. Anagnostopoulos, F. Savva, and P. Triantafillou. Scalable aggregation predictive analytics. *Applied Intelligence*, 48(9):2546–2567, 2018.

[19] C. Anagnostopoulos and P. Triantafillou. Learning set cardinality in distance nearest neighbours. In *2015 IEEE international conference on data mining*, pages 691–696. IEEE, 2015.

[20] C. Anagnostopoulos and P. Triantafillou. Efficient scalable accurate regression queries in in-dbms analytics. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 559–570. IEEE, 2017.

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

[22] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

[23] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1383–1394. ACM, 2015.

[24] M. Atzmueller. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):35–49, 2015.

[25] M. Atzmueller and F. Puppe. Sd-map–a fast algorithm for exhaustive subgroup discovery. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 6–17. Springer, 2006.

[26] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 539–550. ACM, 2003.

[27] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. Macrobase: Analytic monitoring for the internet of things. *arXiv preprint arXiv:1603.00567*, 2016.

[28] A. Belfodil, A. Belfodil, and M. Kaytoue. Anytime subgroup discovery in numerical domains with guarantees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 500–516. Springer, 2018.

[29] Bokeh Development Team. *Bokeh: Python library for interactive visualization*, 2018.

[30] L. Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[31] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 445–456. ACM, 2014.

[32] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)*, 32(2):9, 2007.

[33] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 287–298. ACM, 2004.

[34] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 511–519. ACM, 2017.

[35] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[36] J. Cheney, L. Chiticariu, W.-C. Tan, et al. Provenance in databases: Why, how, and where. *Foundations and Trends® in Databases*, 1(4):379–474, 2009.

[37] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[38] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[39] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[40] 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*, 12(9):1044–1057, 2019.

[41] J. Eckstein, P. L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.

[42] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[43] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *Proceedings of the VLDB Endowment*, 8(1):61–72, 2014.

[44] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[45] R. Foygel Barber, E. J. Candes, A. Ramdas, and R. J. Tibshirani. Predictive inference with the jackknife+. *arXiv preprint arXiv:1905.02928*, 2019.

[46] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[47] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[48] J. H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[49] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[50] J. H. Friedman and N. I. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999.

[51] M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, pages 343–352, 2001.

[52] A. Gepperth and B. Hammer. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*, 2016.

[53] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[54] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[55] O. A. Grigg, V. Farewell, and D. Spiegelhalter. Use of risk-adjusted cusum and rsprtcharts for monitoring in medical contexts. *Statistical methods in medical research*, 12(2):147–170, 2003.

[56] H. Grosskreutz and S. Rüping. On subgroup discovery in numerical domains. *Data mining and knowledge discovery*, 19(2):210–226, 2009.

[57] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1917–1923. ACM, 2015.

[58] A. Guttman and M. Stonebraker. A dynamic index structure for spatial searching. In *Proceedings of the 13th ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1983.

[59] A. Hall, A. Tudorica, F. Buruiana, R. Hofmann, S.-I. Ganceanu, and T. Hofmann. Trading off accuracy for speed in powerdrill. 2016.

[60] G. Hamerly and C. Elkan. Learning the k in k-means. In *Advances in neural information processing systems*, pages 281–288, 2004.

[61] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[62] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[63] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Acm Sigmod Record*, volume 26, pages 171–182. ACM, 1997.

[64] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! *arXiv preprint arXiv:1909.00607*, 2019.

[65] B. Huang, S. Babu, and J. Yang. Cumulon: Optimizing statistical data analysis in the cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM, 2013.

[66] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 277–281. ACM, 2015.

[67] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *Proceedings of the 2016 International Conference on Management of Data*, pages 281–293. ACM, 2016.

[68] B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 841–852. ACM, 2011.

[69] S. Kandula, K. Lee, S. Chaudhuri, and M. Friedman. Experiences with approximating queries in microsoft's production big-data clusters. *Proceedings of the VLDB Endowment*, 12(12):2131–2142, 2019.

[70] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proceedings of the 2016 International Conference on Management of Data*, pages 631–646. ACM, 2016.

[71] 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 *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[72] J. Kennedy. Particle swarm optimization. *Encyclopedia of machine learning, Springer*, pages 760–766, 2010.

[73] N. Khoussainova, M. Balazinska, and D. Suciu. Perfxplain: debugging mapreduce job performance. *Proceedings of the VLDB Endowment*, 5(7):598–609, 2012.

[74] 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.

[75] A. Kipf, D. Vorona, J. Müller, T. Kipf, B. Radke, V. Leis, P. Boncz, T. Neumann, and A. Kemper. Estimating cardinalities with deep sketches. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 1937–1940, New York, NY, USA, 2019. ACM.

[76] R. Koenker and K. F. Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

[77] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504. ACM, 2018.

[78] S. Krishnan and E. Wu. Palm: Machine learning explanations for iterative debugging. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, page 4. ACM, 2017.

[79] K. Krishnanand and D. Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm intelligence*, 3(2):87–124, 2009.

[80] M. Kuhn and K. Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

[81] M. Kulessa, A. Molina, C. Binnig, B. Hilprecht, and K. Kersting. Model-based approximate query processing. *arXiv preprint arXiv:1811.06224*, 2018.

[82] J. Lei, M. GŚell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 2018.

[83] R. J. Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14, 2000.

[84] B. Liu, L.-P. Ku, and W. Hsu. Discovering interesting holes in data. In *IJCAI (2)*, pages 930–935, 1997.

[85] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization & Computer Graphics*, pages 1–1, 2014.

[86] Q. Ma and P. Triantafillou. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1553–1570. ACM, 2019.

[87] N. Mamoulis, S. Bakiras, and P. Kalnis. Evaluation of top-k olap queries using aggregate r–trees. In *International Symposium on Spatial and Temporal Databases*, pages 236–253. Springer, 2005.

[88] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, July 2019.

[89] S. Marsland, J. Shapiro, and U. Nehmzow. A self-organising network that grows when required. *Neural networks*, 15(8-9):1041–1058, 2002.

[90] N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7(Jun):983–999, 2006.

[91] A. Meliou, S. Roy, and D. Suciu. Causality and explanations in databases. *Proceedings of the VLDB Endowment*, 7(13):1715–1716, 2014.

[92] Z. Miao, Q. Zeng, B. Glavic, and S. Roy. Going beyond provenance: Explaining query answers with pattern-based counterbalances. In *Proceedings of the 2019 International Conference on Management of Data*, pages 485–502. ACM, 2019.

[93] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[94] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, Q. Shao, S. Sellamanickam, S. Keerthi, S. Herbert, and S. Dhulipalla. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2029–2038. ACM, 2015.

[95] M. Olma, O. Papapetrou, R. Appuswamy, and A. Ailamaki. Taster: Self-tuning, elastic and online approximate query processing. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 482–493. IEEE, 2019.

[96] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. An empirical analysis of deep learning for cardinality estimation. *arXiv preprint arXiv:1905.06425*, 2019.

[97] H. Papadopoulos, V. Vovk, and A. Gammerman. Regression conformal prediction with nearest neighbours. *Journal of Artificial Intelligence Research*, 40:815–840, 2011.

[98] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1461–1476. ACM, 2018.

[99] Y. Park, A. S. Tajik, M. Cafarella, and B. Mozafari. Database learning: Toward a database that becomes smarter every time. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 587–602. ACM, 2017.

[100] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

[101] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[102] 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*, 12:2825–2830, 2011.

[103] R. A. Poldrack. Region of interest analysis for fMRI. *Social Cognitive and Affective Neuroscience*, 2(1):67–70, 03 2007.

[104] N. Potti and J. M. Patel. Daq: a new paradigm for approximate query processing. *Proceedings of the VLDB Endowment*, 8(9):898–909, 2015.

[105] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[106] Y. Romano, E. Patterson, and E. J. Candès. Conformalized quantile regression. *arXiv preprint arXiv:1905.03222*, 2019.

[107] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[108] S. Roy, L. Orr, and D. Suciu. Explaining query answers with explanation-ready databases. *Proceedings of the VLDB Endowment*, 9(4):348–359, 2015.

[109] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[110] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *International Conference on Extending Database Technology*, pages 168–182. Springer, 1998.

[111] K. Sato. An inside look at google bigquery. *White paper, URL: https://cloud. google. com/files/BigQueryTechnicalWP. pdf*, 2012.

[112] F. Savva. Query-driven learning for next generation predictive modeling & analytics. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1844–1846, 2019.

[113] F. Savva, C. Anagnostopoulos, and P. Triantafillou. Explaining aggregates for exploratory analytics. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 478–487. IEEE, 2018.

[114] F. Savva, C. Anagnostopoulos, and P. Triantafillou. Aggregate query prediction under dynamic workloads. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 671–676. IEEE, 2019.

[115] F. Savva, C. Anagnostopoulos, and P. Triantafillou. Adaptive learning of aggregate analytics under dynamic workloads. *Future Generation Computer Systems*, 2020.

[116] F. Savva, C. Anagnostopoulos, and P. Triantafillou. Ml-aqp: Query-driven approximate query processing based on machine learning. *arXiv preprint arXiv:2003.06613*, 2020.

[117] F. Savva, C. Anagnostopoulos, and P. Triantafillou. Surf: identification of interesting data regions with surrogate models. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1321–1332. IEEE, 2020.

[118] F. Savva, C. Anagnostopoulos, P. Triantafillou, and K. Kolomvatsos. Large-scale data exploration using explanatory regression functions. *ACM Transactions on Knowledge Discovery from Data*, 2020.

[119] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015.

[120] K. Sequeira and M. Zaki. Schism: A new approach for interesting subspace mining. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 186–193, 2004.

[121] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[122] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

[123] L. Sidirourgos, P. Boncz, M. Kersten, et al. Sciborq: Scientific data management with bounds on runtime and quality. 2011.

[124] B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[125] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[126] I. Steinwart, A. Christmann, et al. Estimating conditional quantiles with the help of the pinball loss. *Bernoulli*, 17(1):211–225, 2011.

[127] J. Sun and G. Li. An end-to-end learning-based cost estimator. *Proceedings of the VLDB Endowment*, 13(3):307–319, 2019.

[128] A. S. Szalay, J. Gray, A. R. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg. The sdss skyserver: public access to the sloan digital sky server data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 570–581. ACM, 2002.

[129] I. Takeuchi, Q. V. Le, T. D. Sears, and A. J. Smola. Nonparametric quantile estimation. *Journal of machine learning research*, 7(Jul):1231–1264, 2006.

[130] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing using deep generative models. *arXiv preprint arXiv:1903.10000*, 2019.

[131] A. Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

[132] M. R. Uddin, C. Ravishankar, and V. J. Tsotras. Finding regions of interest from trajectory data. In *2011 IEEE 12th MDM International Conference on Mobile Data Management*, volume 1, pages 39–48, 2011.

[133] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1009–1024. ACM, 2017.

[134] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1231–1245. ACM, 2015.

[135] X. Wang, A. Meliou, and E. Wu. Qfix: Diagnosing errors through query histories. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1369–1384. ACM, 2017.

[136] Z. Wang, D. Cashman, M. Li, J. Li, M. Berger, J. A. Levine, R. Chang, and C. Scheidegger. Neuralcubes: Deep representations for visual data exploration. *arXiv preprint arXiv:1808.08983*, 2018.

[137] A. Wasay, X. Wei, N. Dayan, and S. Idreos. Data canopy: Accelerating exploratory statistical analysis. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 557–572. ACM, 2017.

[138] L. Woltmann, C. Hartmann, M. Thiele, D. Habich, and W. Lehner. Cardinality estimation with local deep learning models. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–8, 2019.

[139] K.-C. Wong. Evolutionary multimodal optimization: A short survey. *arXiv preprint arXiv:1508.00457*, 2015.

[140] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8):553–564, 2013.

[141] E. Wu, S. Madden, and M. Stonebraker. Subzero: A fine-grained lineage system for scientific databases. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 865–876. IEEE, 2013.

[142] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous sampling for online aggregation over multiple queries. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 651–662. ACM, 2010.

[143] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Selectivity estimation with deep likelihood models. *arXiv preprint arXiv:1905.04278*, 2019.

[144] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[145] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-ola: Generalized online aggregation for interactive analysis on big data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 913–918. ACM, 2015.

[146] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 277–288. ACM, 2014.