Fast Data Analytics by Learning

by

Yongjoo Park

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2017

Yongjoo Park

pyongjoo@umich.edu

ORCID iD: 0000-0003-3786-6214

To my mother

Acknowledgements

I thank my co-advisors, Michael Cafarella and Barzan Mozafari. I am deeply grateful that I could start my graduate studies with Mike Cafarella. Mike was the one of the nicest persons I have met (I can say, throughout my life). He also deeply cared of my personal life as well as my graduate study. He always encouraged and supported me so I could make the best decisions for my life. He was certainly more than a simple academic advisor. Mike also has excellent talents in presentation. His descriptions (on any subject) are well-organized and straightforward. Sometimes, I find myself trying to imitate his presentation styles when I give my own presentations, which, I believe, is natural since I have been advised by him for more than five years now.

I started work with Barzan after a few years since I came to Michigan. He is excellent in writing research papers and presenting work in the most interesting, formal, and concise way. He devoted tremendous amount of time for me. I learned great amount of writing and presentation skills from him. I am glad I will be his first PhD student, and I hope he is glad about the fact as well. Now as I finish my long (and much fun) PhD study, I deeply understand the importance of having good advisors, and I am sincerely grateful that I was advised by Mike and Barzan. I always think my life has been full of fortunes I cannot explain, and having them as my advisors was one of them.

I am sincerely grateful of my thesis committee members. Both Carl Lagoze and Eytan Adar were kind enough to serve on my thesis committee, and they were supportive. Eytan Adar gave detailed feedbacks on this dissertation, which helped me improve its completeness greatly. H.V. Jagadish gave me wonderful advice on presentations and writing. Even from my relatively short experience with him, I could immediately sense that he is wise and is a great mentor.

My PhD study was greatly helped by the members of the database group at the University of Michigan, Ann Arbor. Dolan Antenucci helped me setting up systems and shared with me many interesting real-world datasets. Some of the results in this dissertation are based on the datasets provided by him. I also had good discussions with Mike Anderson, Shirley Zhe Chen, and Matthew Burgess. I regret that I did not have more collaborations with the people at the database group.

My graduate study was not possible without the financial support from Jeongsong Cultural Foundation. When I first came to the United States as a Masters student (without guaranteed funding), Jeongsong Cultural Foundation provided me with tuition support. I always thank it, and I am going to pay my debt to the society. Also, Kwanjeong Educational Foundation supported me for my Phd studies. With its help, I could only focus on my research without any financial concerns. I will be someone like them, who can contribute to the society. My research is one part of such efforts, but I hope I can do more.

Last, but most importantly, I deeply thank my parents and my sister who always supported me and my study. I was fortunate enough to have my family. I have always been a taker. I should mention my mother, my beloved mother, who devoted her entire life to her children. My life has become better and better, over time. Now I do not have to worry too much about paying phone bills or bus fees. Now I can easily afford Starbucks coffee and commute by drive. But I want to go back, to the time, when I was young. I miss my mother. I miss the time when I fell asleep by her, watching a late night movie on television. I miss the food she made for me, I miss the conversations with her. I miss her smile, and I miss that there always was a person in this world who would stand and speak for me however bad things I did. I received the best education in the world only due to her selfless and stupid amount of sacrifice. She only wanted her children to receive the best support, have the most tasty food, and enjoy every thing other kids would enjoy. Yet, she was foolish enough to never enjoy any such. To her, raising her kids, me and my sister, was her only happiness, and she conducted the task better than anyone. Both of her children grew up wonderfully. I conducted fine research during my graduate studies, and I will continue to do so. My sister became a medical doctor. I want to tell my mom that her life was worth more than any, and she should be proud of herself.

Contents

List of Figures

List of Tables

Abstract

Fast Data Analytics by Learning

by

Yongjoo Park

Chairs: Michael Cafarella and Barzan Mozafari

Today, we collect a large amount of data, and the volume of the data we collect is projected to grow faster than the growth of the computational power. This rapid growth of data inevitably increases query latencies, and horizontal scaling alone is not sufficient for real-time data analytics of big data. Approximate query processing (AQP) speeds up data analytics at the cost of small quality losses in query answers. AQP produces query answers based on *synopses* of the original data. The sizes of the synopses are smaller than the original data; thus, AQP requires less computational efforts for producing query answers, thus can produce answers more quickly. In AQP, there is a general tradeoff between query latencies and the quality of query answers; obtaining higher-quality answers requires longer query latencies.

In this dissertation, we show *we can speed up the approximate query processing without reducing the quality of the query answers by optimizing the synopses using two approaches.* The two approaches we employ for optimizing the synopses are as follows:

1. **Exploiting past computations:** We exploit the answers to the past queries. This approach relies on the fact that, if two aggregation involve common or correlated values, the aggregated results must also be correlated. We formally capture this idea using a probabilistic distribution function, which is then used to refine the answers to new queries.

2. **Building task-aware synopses:** By optimizing synopses for a few common types of data analytics, we can produce higher quality answers (or more quickly for certain target quality) to those data analytics tasks. We use this approach for constructing synopses optimized for searching and visualizations.

For exploiting past computations and building task-aware synopses, our work incorporates statistical inference and optimization techniques. The contributions in this dissertation resulted in up to 20$\times$ speedups for real-world data analytics workloads.

Chapter 1

Introduction

We collect huge amount of data. In 2015, one billion Facebook users generated 31 million messages every minute. YouTube stores 300 hours of new videos every minute. Walmart collects 40 petabytes of data a day. Boeing generates one terabyte of data for every flight. The prevalence of portable 4K video recording devices, high-frequency sensors, and dash cams accelerate this trend.

Analyzing data is crucial for business; and we want to analyze them quickly, in real-time. Data-driven decision making brings 5%-6% higher productivity in 179 large publicly traded firms [34]. However, slow query responses (longer than 500 milliseconds–three seconds) in data analytics systems significantly reduce the productivity in data exploration tasks and the amount of data we explore [114, 157].

The rapid growth in the volume of data makes real-time data analytics hard. An analyst might be able to reduce query latencies by a factor of 100 by horizontal scaling [159, 181, 182]. However, this simply means she has to spend as least 100 times as much money as before for setting up the machines; also, achieving real-time data analytics by horizontal scaling is only possible when her spending increases faster than the growth of data, which is not economic.

This phenomenon means essentially that, for providing real-time data analytics at reasonable costs, we must reduce the amount of data we process. The popular approach in the literature is answering queries using *synopses*, instead of the original raw data [10, 12, 15, 16, 38, 58, 62, 101, 184]. Figure 1.1 depicts a general architecture of those systems. In most cases, the synopses are constructed offline. When a query is issued, data analytics systems answer the query only using the query synopses. Typically, the sizes of the synopses are much smaller than the original data; thus, computing the query answers using synopses is much faster.

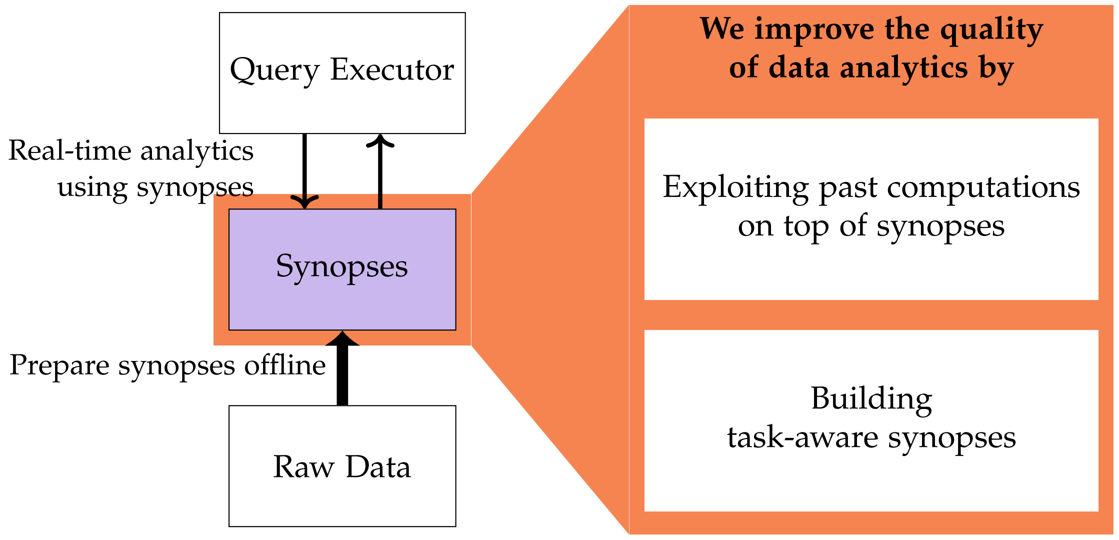

Query Executor
Real-time analytics using synopses
Synopses
Prepare synopses offline
Raw Data
We improve the quality of data analytics by
Exploiting past computations on top of synopses
Building task-aware synopses

Figure 1.1: Architecture of data analytics systems that rely on *synopses* for real-time data analytics (left), and our contributions in this dissertation (right). In this dissertation, we further speed up data analytics (i) by exploiting past computations on top of existing synopses, and (ii) by building task-aware synopses.

Note that the sizes of synopses have a direct impact on the quality of answers and the query latencies. That is, if the sizes of query synopses are small, we can answer queries fast; however, the quality of the answers will be low. On the other hand, if the sizes of the query synopses are large, it takes longer to answer the same queries; however, the quality of the answers will be high. This means there is an important tradeoff between the query latency and the quality of the query answers. Often, data analytics systems provide users with the options to place upper-bounds on query processing times or lower-bounds on the quality of query answers. However, the tradeoff between the query latency and the quality of the query answers remains the same.

In this dissertation, we show *we can speed up the query processing without reducing the quality of the query answers.* We achieve this by improving the average quality of query answers given the fixed space budget for the synopses. For improving the quality of query answers, we employ two approaches. First, *we exploit the past computations.* Exploiting past computations helps us avoid processing the same data repeatedly if a new query involves common (or correlated) data as the ones processed in the past. Second, *we optimize synopses for specific tasks.* Optimizing synopses for specific tasks help data analytics systems bring higher quality answers when they process the same amount of data.

We apply the first approach—exploiting the past computations—for faster data aggregations, and apply the second approach—task-specific synopses—for faster data mining and visualizations. In section 1.1, we provide the reason why we take those two approaches for speeding up those three specific data analytics tasks. Figure 1.1 summarizes key technical insights in applying our approaches to those three data analytics tasks.

1.1 Data Analytics Tasks and Synopses Construction

In this section, we explore popular data analytics tasks and the possible approaches for building synopses.

1.1.1 Data Analytics Tasks

To understand the popular big data analytics tasks commonly used/needed in the industry, we referred to the survey conducted in 2011 [147]. The survey was based on the 325 respondents who identified themselves as academics or vendor employees. Table 1.1 summarizes the data analytics tasks used in the organizations of those respondents.

Table 1.1: Popular Data Analytics Tasks

Task	Subtasks
data visualization	• **real-time visualizations** • visualizations of diverse types (e.g., hierarchies and neural nets)
data mining	• text mining • **predictive analytics**
data transformation	• extracting a fact table • transformation to structured data
OLAP	• **basic aggregations** • statistical analysis
complex event processing	• stream data processing

In this dissertation, we focus on applying our two approaches—past computation exploitation and task-aware synopses—for speeding up three fundamental subtasks. We mark those three subtasks in bold in Table 1.1.

1.1.2 Approaches to Building Synopses

We identify two orthogonal aspects in building synopses: task-sensitivity and online/offline synopses construction. The second aspect—online/offline synopses construction—captures the relative computational costs spent before queries come (offline) to the computational costs spent as processing queries (online). Using these two aspects, we categorize the two approaches we use in this dissertation. This analysis also suggests a new opportunity.

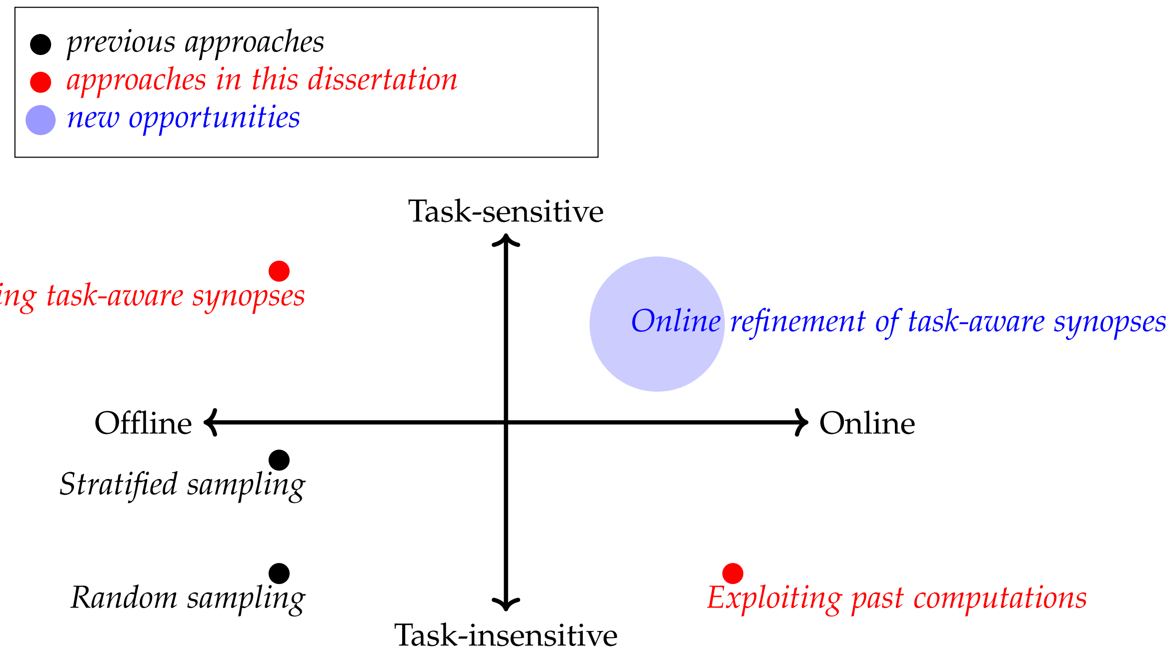

● *previous approaches*
● *approaches in this dissertation*
● *new opportunities*
Task-sensitive
Building task-aware synopses
Online refinement of task-aware synopses
Offline
Online
Stratified sampling
Random sampling
Exploiting past computations
Task-insensitive

Figure 1.2: Two orthogonal aspects in building synopses.

Figure 1.2 depicts the aspects of our approaches. Building task-aware synopses are optimized for target data analytics tasks while its synopses constructions are fully offline (i.e., before queries come). Exploiting past computations refine synopses online but it does not necessarily optimize the underlying synopses for target tasks. In fact, the approach of exploiting past computations should be able to combined with building task-aware synopses. Although this dissertation does not include such an approach, more advanced synopses in the future would combine both approaches so that they can refine the synopses online in a task-aware manner. We mark this new opportunity in blue in Figure 1.2.

1.2 Overview and Contributions

This dissertation is organized into two parts. In the first part, we present the methods that reduce query latencies by exploiting the past computations. In the second part, we present the methods that reduce query latencies by optimizing synopses for specific tasks. Table 1.2 summarizes the motivations behind our techniques.

Table 1.2: Dissertation Overview

Part	Motivation	Chapter
Exploiting past computations	**Aggregation:** The models we learn from the answers to the past queries can be used to avoid computing raw data for new queries.	3
Task-aware synopses	**Predictive analytics:** kNN-aware hash functions can be more efficient for nearest-neighbor problems compared to randomly-generated hash functions.	4
	Visualization: By selecting a subset of data items that are optimal for visualization purposes, we can produce higher-quality visualizations compared to random samples.	5

Note that finding k-most similar items is the key algorithm for k-nearest neighbor classifiers and collaborative filtering.

The contributions in this dissertation have also been presented at Computer Science conferences and workshops, including the 2017 ACM SIGMOD conference [139], the 42nd International Conference on Very Large Data Bases 2016 [137], and the International Conference on Data Engineering 2016 [138].

1.2.1 Exploiting Past Computations

Fast Aggregation by Learning form Past Query Answers

In today's databases, previous query answers rarely benefit answering future queries. We change this paradigm in an approximate query processing (AQP) context. We make the following observation: the answer to each query reveals some degree of knowledge about the answer to another query because their answers stem from the same underlying distribution that has produced the entire dataset. Exploiting and refining this knowledge

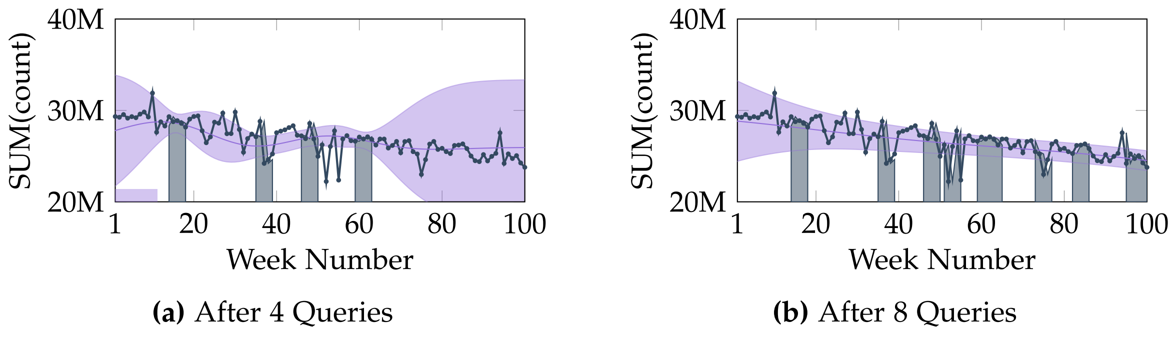


Figure 1.3: Examples of models built after processing (a) 4 queries, and (b) 8 queries. Processed queries are represented by gray boxes. These models can be used to speed up future query processing.

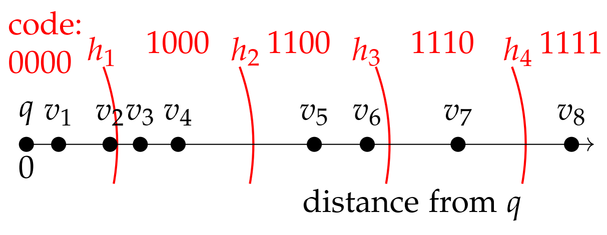
should allow us to answer queries more analytically, rather than by reading enormous amounts of raw data. Also, processing more queries should continuously enhance our knowledge of the underlying distribution, and hence lead to increasingly faster response times for future queries.

In Figure 3.1, we visualize this idea using a real-world Twitter dataset. Here, our technique learns a model for the number of occurrences of certain word patterns (known as *n-grams*, e.g., "bought a car") in tweets. Figure 1.3(a) shows this model (in purple) based on the answers to the first two queries asking about the number of occurrences of these patterns, each over a different time range. Since the model is probabilistic, its 95% confidence interval is also shown (the shaded area around the best current estimate). As shown in Figure 1.3(b), DBL further refines its model as more new queries are answered.

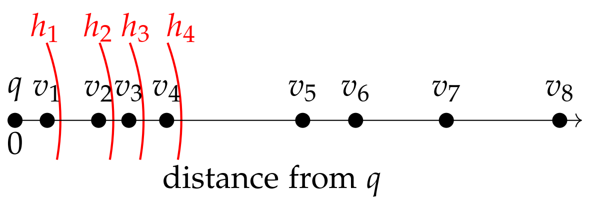
We call this novel idea—learning from past query answers—Database Learning. We exploit the principle of maximum entropy to produce answers, which are in expectation guaranteed to be more accurate than existing sample-based approximations. Empowered by this idea, we build a query engine on top of Spark SQL, called Verdict. We conduct extensive experiments on real-world query traces from a large customer of a major database vendor. Our results demonstrate that Verdict supports 73.7% of these queries, speeding them up by up to 23.0× for the same accuracy level compared to existing AQP systems

1.2.2 Task-Aware Synopses

Fast Searching by Neighbor-Sensitive Hashing



(a) Random Hash Functions



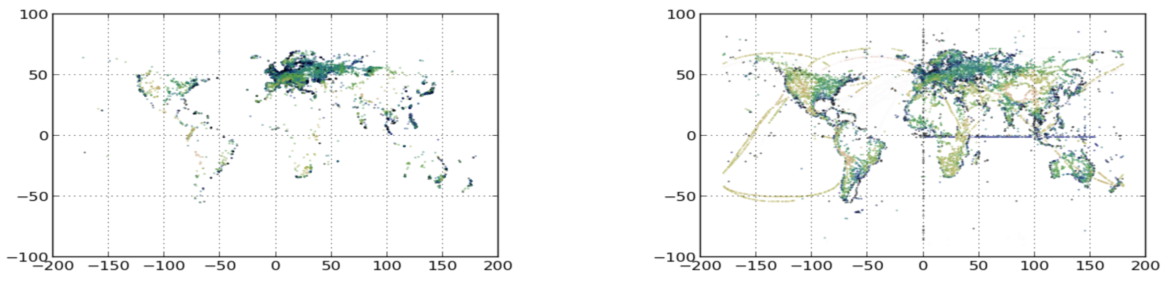
(b) Our Hash Functions

Figure 1.4: An example that kNN-oriented hashing functions (on the right) can bring higher search accuracy compared to using randomly generated hash functions (on the left).

Approximate kNN (k-nearest neighbor) techniques using binary hash functions are among the most commonly used approaches for overcoming the prohibitive cost of performing exact kNN queries. However, the success of these techniques largely depends on their hash functions' ability to distinguish kNN items; that is, the kNN items retrieved based on data items' hashcodes, should include as many true kNN items as possible. A widely-adopted principle for this process is to ensure that similar items are assigned to the same hashcode so that the items with the hashcodes similar to a query's hashcode are likely to be true neighbors.

In this work, we abandon this heavily-utilized principle and pursue the opposite direction for generating more effective hash functions for kNN tasks. That is, we aim to increase the distance between similar items in the hashcode space, instead of reducing it. We achieve this by placing more hash functions between close (neighbor) data items as depicted in Figure 1.4, which makes those neighbor items more distinguishable based on the hashcodes.

Our contribution begins by providing theoretical analysis on why this seemingly counter-intuitive approach leads to a more accurate identification of kNN items. Our analysis is followed by a proposal for a hashing algorithm that embeds this novel principle. Our empirical studies confirm that a hashing algorithm based on this counter-intuitive idea significantly improves the efficiency and accuracy of state-of-the-art techniques.



(a) Random Sample **(b)** Our Sample

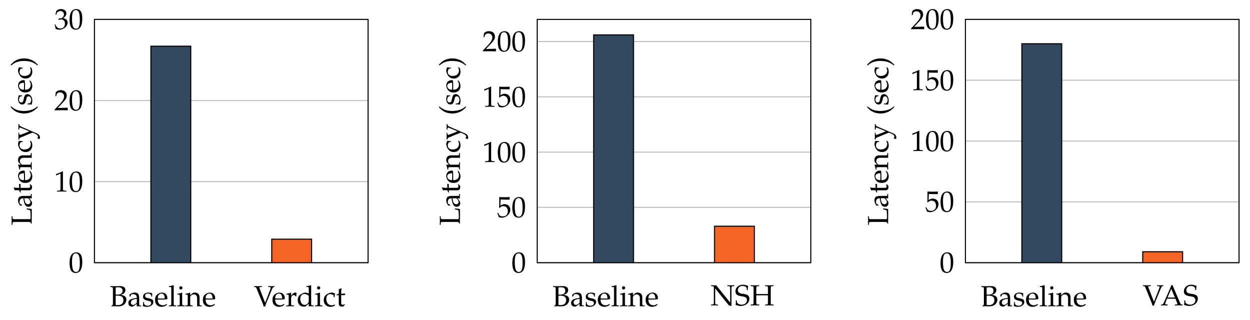
Figure 1.5: An example that visualization-aware sampling can bring higher quality visualizations compared to visualizing randomly chosen data items.

Interactive visualizations are crucial in ad-hoc data exploration and analysis. However, with the growing number of massive datasets, generating visualizations in interactive timescales is increasingly challenging. One approach for improving the speed of the visualization tool is via data reduction. However, common data reduction techniques, such as uniform and stratified sampling, do not exploit the fact that the sampled tuples will be transformed into a visualization for human consumption.

We depict the issues of uniform random sampling in Figure 1.5(a). The figure visualizes ten thousands data points. Since the sample is generated by sampling data points with uniform probabilities, the sample tends to include more points from denser areas. However, for the visualization purpose, because many data points overlap, they do not provide good visualization quality.

In this work, we develop a visualization-aware sampling (VAS) for high quality visualizations. The key to our sampling method's success is in choosing a set of tuples that minimizes a visualization-inspired loss function. Our user study confirms that our proposed loss function correlates strongly with user success in using the resulting visualizations. As depicted in Figure 1.5(b), the sample by VAS brings a higher-quality visualization. Our experiments show that (i) VAS indeed improves user's success by up to 35% in various visualization tasks, and (ii) VAS can achieve a required visualization quality up to 400× faster.

1.2.3 Summary of Results



(a) Verdict **(b)** NSH **(c)** VAS

Figure 1.6: Performance improvements achieved by our proposed techniques.

In this section, we summarize the query speedup (or latency reductions) achieved by each of our techniques presented in this dissertation. More details on our empirical studies are described in the main body of this dissertation.

First, we report the speedup by Verdict, the approximate query processing system that exploits a model built on the answers to past queries for speeding on future query processing. Figure 1.6(a) compares two systems' query latencies, where the two systems are an existing approximate query processing system and the Verdict system implemented on top of the existing approximate query processing system. When the target error bound was 4%, Verdict could return query answers 9.3× faster on average.

Second, we report the speedup by NSH, the approximate searching technique that rely on hash functions specifically generated for *k*NN tasks. Figure 1.6(b) compares the two methods' query latencies, where the two systems are locality-sensitive hashing (a popular hashing-based *k*NN method) and our method (NSH). When the target recall 40%, NSH could finish searching 6.24× faster on average.

Third, we report the speedup by VAS, a sampling algorithm for approximate visualizations; the algorithm chooses a subset of data that maximizes the visualization quality. Figure 1.6(b) compares the two methods' query latencies, where the two methods are uniform random sampling and VAS. For a certain target visualization quality metric, VAS could use a much smaller sample compared to uniform random sampling, which resulted in 20× speedups on average.

Chapter 2

Background

In this chapter, we briefly review modern techniques for large-scale data analytics systems. In the first section, we present several techniques developed for *exact* large-scale data analytics. This first section shows that those techniques are orthogonal to approximate query processing (AQP). In the second section, we introduce existing techniques for AQP. This second section also discusses several issues in those existing techniques.

2.1 Techniques for Exact Data Analytics

We present three techniques for speeding up large-scale data analytics: horizontal scaling, columnar databases, and in-memory databases. Note that these techniques are not exclusive one another. Thus, a system can employ all those three techniques. In contrast to AQP, these techniques do not involve approximations for speeding up query processing.

2.1.1 Horizontal Scaling

Apache Hadoop [2] and Apache Spark [5] aim to provide linear scalability through the "embarrassingly parallel" map-reduce operations. Conceptually, the map operations transform each item in a list to another form, and the reduce operations aggregate those transformed items. High scalability is achieved by enforcing those map and reduce operations not to share any global states; thus, there does not exist any locks that impede concurrent executions.

Recently, systems that provide convenient SQL interface have been developed on top of those highly scalable systems. The examples include Apache Hive [3], Spark SQL, and Apache Impala (incubating) [4]. AQP systems can also be implemented on top of those horizontally scalable systems; thus, the advances in horizontally scalable systems also benefit AQP.

2.1.2 Columnar Databases

Many conventional relational database systems store data row by row. However, as interests on data analytics increase, column-oriented storage layouts appeared. Database systems with columnar layouts provide faster processing of analytic workloads by reading only necessary columns [91, 143] Also, columnar format can achieve good compression [105]. Performing data analytics operations directly on compressed data can also reduce I/O costs. AQP systems can also employ columnar format.

2.1.3 In-memory Databases

Keeping the entire data in memory brings faster accesses and real-time analytics. Many commercial database vendors introduced in-memory database processing to support large-scale applications [53, 87, 94, 133, 156, 183, 183]. In practice, however, storing the entire data in-memory can be extremely costly. Also, achieving real-time query latency requires a cluster that consists of many number of workers, which also increases the overall costs. AQP can greatly reduce the cost for in-memory databases, since the sizes of samples are much smaller than the original data; thus, they can more easily fit in memory.

2.2 Approaches for Approximate Query Processing

In this section, we review two approaches for AQP: random sampling and random projections. Random sampling reduces the number of the data items by obtaining a sample of the original data. In contrast, random projections converts the data into special representations so that data analytics systems can exploit some properties of the new representations. Both approaches lose information in the sampling or projection process; as a result, answering queries using those samples or based on new representations produce approximate answers.

2.2.1 Random Sampling

Using a sample of an original data is a popular approach for estimating the results of aggregations and for visualizing large-scale datasets. Processing the reduced amount of data (in a sample) can bring significant speedups when the sample size is much smaller than the size of the original data.

For aggregations, *the quality of approximate answers only relies on the sample size, not the ratio between the sample size and the original dataset size.* This makes the sampling-based AQP much appealing since the quality of approximate answers do not increase even if the size of the original dataset increases. In other words, AQP can still produce high-quality answers (within the same time-bounds) even if the size of the original data grows.

Uniform Random Sampling— The most basic technique for random sampling is uniform random sampling. Suppose there exists N numeric values in the original data. Uniform random sampling obtains a sample (say, n values out of those N values) by sampling values uniformly at random from the original dataset. The sample size n is determined by the data analytics systems considering requested query latencies and error bounds.

According to the central limit theorem, a sample mean follows a normal distribution with its mean being equal to the population mean and its variance being equal to the population variance divided by the sample size n. Thus, as the sample size (i.e., n) increases, the shape of normal distribution becomes narrower being centered at the population mean. As a result, the sample mean becomes more accurate.

Based on this result of the central limit theorem, the range within which the population mean will exist with high probability can also be inferred. Many AQP systems rely on this property to estimate the population mean and its confidence intervals. The central limit theorem can also be applied for estimating other population statistics, such as the sum of the values in the original data, the fraction of the items that satisfy certain conditions, etc. Relying on those results, AQP systems can approximately answer other aggregate functions, such as sum, count, etc., based on a uniform random sample.

One issue of the uniform random sampling is that rare subpopulations are likely to be under-represented in a sample. This causes an issue when the user wants to analyze rare subpopulations.

Stratified Sampling— Stratified sampling divides the original data into multiple strata and performs independent uniform random sampling within each stratum. Thus, stratified sampling prevents a certain group (i.e., stratum) from being under-represented, which was an issue of the uniform random sampling. Stratified sampling is useful when the rare subpopulations the user wants to analyze coincides with one of those strata based on which the stratified sampling is performed.

Stratified sampling does not completely address the issue of uniform random sampling, since within each stratum, still uniform random sampling is performed. Therefore, the issue of uniform random sampling may still exist within each stratum.

2.2.2 Random Projections

Let each of original data items be represented by a multidimensional vector whose elements are real-values. Random projections transform those multidimensional vectors into their respective bit vectors (of a certain predetermined length). The bit vectors approximately preserve the relationships among the original data items, such as Euclidean distance or inner projects. As a result, some data analytics tasks that rely on the distances among the data items, e.g., retrieving similar items to a given query item, can be performed based on the transformed representations. Since the random projections preserve the relationship among the original data items only *approximately*, the data analytics tasks performed based on those transformed representations are also approximate.

Retrieving similar items to a given query item can be much faster when performed based on the transformed bit vectors. This is because those bit vectors can easily be stored in a hash table, and the items that have the similar bit vectors to the query item can be then retrieved using lookup operations in the hash table. Here, the similarity between two bit vectors is measured by counting the number of the positions that contain different bits between those two bit vectors. One can understand this process of transforming original data items into bit vectors and storing them in a hash table as the process of assigning the data items into buckets; the assignment process tends to preserve the distances among the data items.

In general, random projections may not be an optimal choice for two reasons. First, random projections do not consider density of data items. For instance, there could exist some clusters within which many data items are located. Random projections that do not consider such data distributions cannot effectively distinguish the data items that belong to the same cluster. Second, random projections do not consider target data analytics tasks. For instance, random projections can be used whether the user is interested in retrieving only a small number of similar items or the user is interested in approximately sorting all data items according to the distance from a query item. This generality brings inefficiency when the user is only interested in a small number of similar items.

Part I
Exploiting Past Computations

Chapter 3

Faster Data Aggregations by Learning from the Past

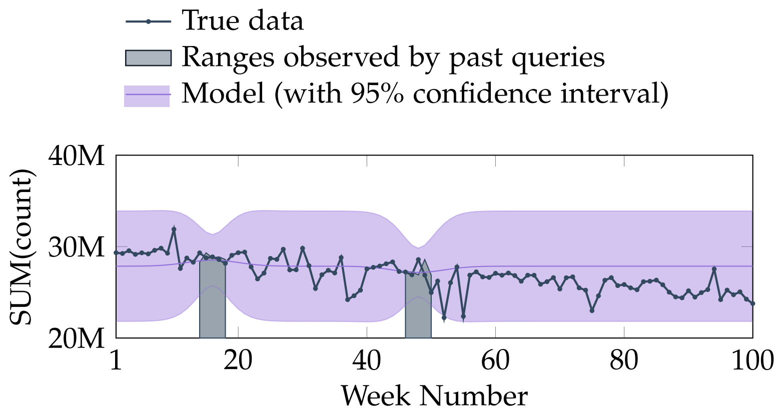
This dissertation presents two approaches—exploiting past computations and building task-aware synopses—for improving the quality of approximate query processing (AQP). In this chapter, we present a technique that exploits past computations for higher-quality AQP. This technique speeds up approximate aggregations when target error bounds are fixed.

3.1 Motivation

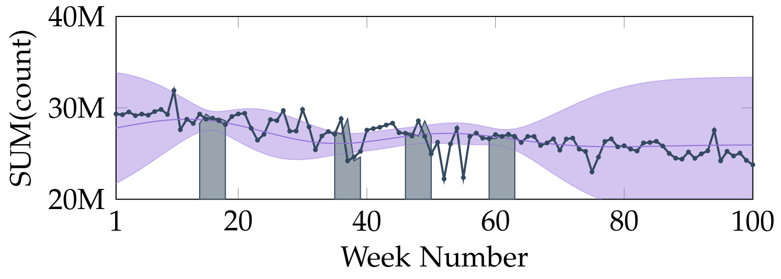
In today's databases, the answer to a previous query is rarely used for speeding up new queries. Besides a few limited benefits (see *Previous Approaches* below), the work (both I/O and computation) performed for answering past queries is often wasted afterwards. However, in an approximate query processing context (e.g., [15, 38, 68, 71, 144, 185]), one might be able to change this paradigm and reuse much of the previous work done by the database system based on the following observation:

> *The answer to each query reveals some* fuzzy knowledge *about the answers to other queries, even if each query accesses a different subset of tuples and columns.*

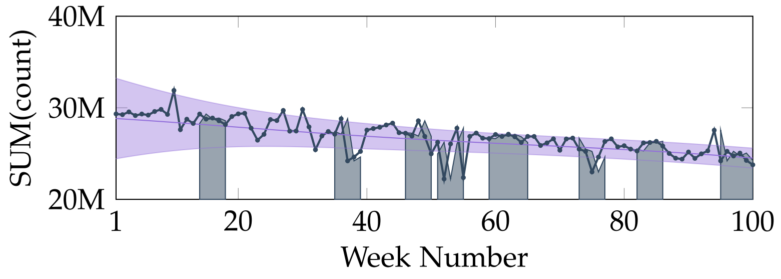
This is because the answers to different queries stem from the same (unknown) underlying distribution that has generated the entire dataset. In other words, each answer reveals a piece of information about this underlying but **unknown distribution**. Note that having a concise statistical model of the underlying data can have significant performance benefits. In the ideal case, if we had access to an incredibly precise model of the underlying data, we would no longer have to access the data itself. In other words,



(a) After 2 Queries



(b) After 4 Queries



(c) After 8 Queries

Figure 3.1: An example of how database learning might continuously refine its model as more queries are processed: after processing (a) 2 queries, (b) 4 queries, and (c) 8 queries. We could deliver more accurate answers if we combined this model with the approximate answers produced by traditional sampling techniques.

we could answer queries more efficiently by analytically evaluating them on our concise model, which would mean reading and manipulating a few kilobytes of model parameters rather than terabytes of raw data. While we may never have a perfect model in practice, even an imperfect model can be quite useful. Instead of using the entire data (or even a large sample of it), one can use a small sample of it to quickly produce a rough approximate answer, which can then be calibrated and combined with the model to obtain a more accurate approximate answer to the query. **The more precise our model, the less need for actual data, the smaller our sample, and consequently, the faster our response time.** In particular, if we could somehow continuously improve our model—say, by *learning* a bit of information from every query and its answer—we should be able to **answer new queries using increasingly smaller portions of data, i.e., become smarter and faster as we process more queries.**

We call the above goal *Database Learning* (DBL), as it is reminiscent of the inferential goal of Machine Leaning (ML) whereby past observations are used to improve future predictions [35, 36, 150]. Likewise, our goal in DBL is to enable a similar principle by **learning from past observations, but in a query processing setting**. Specifically, in DBL, we plan to treat approximate answers to past queries as observations, and use them to refine our posterior knowledge of the underlying data, which in turn can be used to speed up future queries.

In Figure 3.1, we visualize this idea using a real-world Twitter dataset [17, 19]. Here, DBL learns a model for the number of occurrences of certain word patterns (known as *n-grams*, e.g., "bought a car") in tweets. Figure 3.1(a) shows this model (in purple) based on the answers to the first two queries asking about the number of occurrences of these patterns, each over a different time range. Since the model is probabilistic, its 95% confidence interval is also shown (the shaded area around the best current estimate). As shown in Figure 3.1(b) and Figure 3.1(c), DBL further refines its model as more new queries are answered. This approach would allow a DBL-enabled query engine to provide increasingly more accurate estimates, *even for those ranges that have never been accessed by previous queries*—this is possible because DBL finds the most likely model of the entire area that fits with the past query answers. The goal of this simplified example[1] is to illustrate the possibility of (i) significantly faster response times by processing smaller samples of the data for the same answer quality, or (ii) increasingly more accurate answers for the same sample size and response time.

Challenges— To realize DBL's vision in practice, three key challenges must be overcome. First, there is a *query generality* challenge. DBL must be able to transform a wide class of SQL queries into appropriate mathematical representations so that they can be fed into statistical models and used for improving the accuracies of new queries. Second, there is a *data generality* challenge. To support arbitrary datasets, DBL must not make any

[1]In general, DBL does not make any *a prior* assumptions regarding correlations (or smoothness) in the data; any correlations present in the data will be naturally revealed through analyzing the answers to past queries, in which case DBL automatically identifies and makes use of them.

assumptions about the data distribution; the only valid knowledge must come from past queries and their respective answers. Finally, there is an *efficiency* challenge. DBL needs to strike a balance between the computational complexity of its inference and its ability to reduce the error of query answers. In other words, DBL needs to be both effective and practical.

Our Approach— Our vision of database learning (DBL) [125] might be achieved in different ways depending on the design decisions made in terms of query generality, data generality, and efficiency. In this chapter, we present our prototype system that solves those challenges for a certain class of analytic SQL queries; we call the system Verdict.

From a high-level, Verdict addresses the three challenges—query generality, data generality, and efficiency—as follows. First, Verdict supports SQL queries by decomposing them into simpler atomic units, called *snippets*. The answer to a snippet is a single scalar value; thus, our belief on the answer to each snippet can be expressed as a random variable, which can then be used in our mathematical model. Second, to achieve data generality, Verdict employs a *non-parametric* probabilistic model, which is capable of representing arbitrary underlying distributions. This model is based on a simple intuition: *when two queries share some tuples in their aggregations, their answers must be correlated.* Our probabilistic model is a formal generalization of this idea using *the principle of maximum entropy* [162]. Third, to ensure computational efficiency, we keep our probabilistic model in an analytic form. At query time, we only require a matrix-vector multiplication; thus, the overhead is negligible.

Contributions— This chapter presents the following contributions:

1. We introduce the novel concept of *database learning* (DBL). By learning from past query answers, DBL allows DBMS to continuously become smarter and faster at answering new queries.
2. We provide a concrete instantiation of DBL, called Verdict. Verdict's strategies cover 63.6% of TPC-H queries and 73.7% of a real-world query trace from a leading vendor of analytical DBMS. Formally, we also prove that Verdict's expected errors are never larger than those of existing AQP techniques.
3. We integrate Verdict on top of Spark SQL, and conduct experiments using both benchmark and real-world query traces. Verdict delivers up to 23$\times$ speedup and 90% error reduction compared to AQP engines that do not use DBL.

The rest of this chapter is organized as follows. Section 4.2 overviews Verdict's workflow, supported query types, and internal query representations. Sections 3.3 and 3.4 describe the internals of Verdict in detail, and section 3.7 presents Verdict's formal guar-

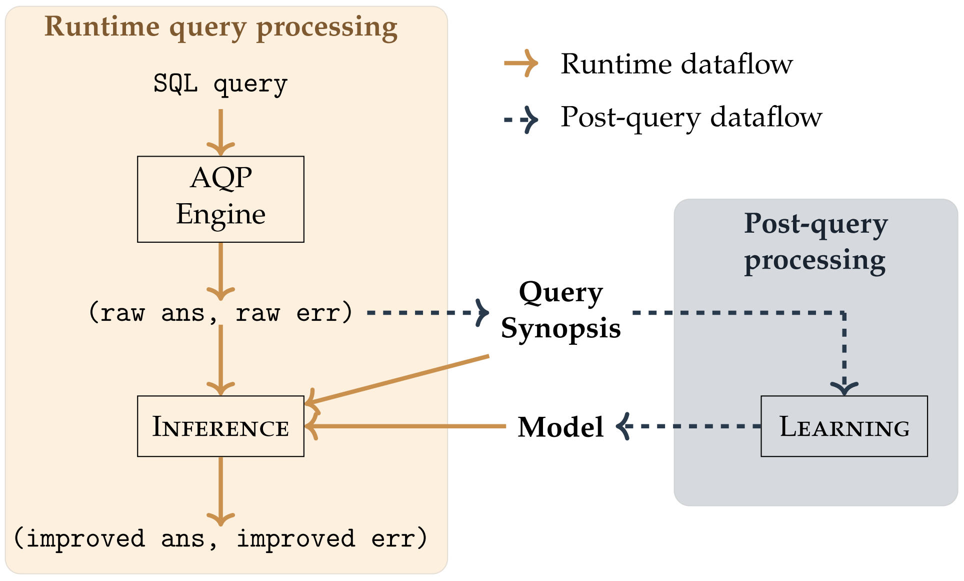


Figure 3.2: Workflow in Verdict. At query time, the INFERENCE module uses the *Query Synopsis* and the *Model* to improve the query answer and error computed by the underlying AQP engine (i.e., *raw answer/error*) before returning them to the user. Each time a query is processed, the raw answer and error are added to the *Query Synopsis*. The LEARNING module uses this updated *Query Synopsis* to refine the current *Model* accordingly.

antees. Section 3.5 summarizes Verdict's online and offline processes, and Section 3.6 discusses Verdict's deployment scenarios. section 5.6 reports our empirical results. Section 4.5 discusses related work, and section 5.8 concludes the chapter with future work.

3.2 Verdict Overview

In this section, we overview the system we have built based on DBL, called Verdict. Section 3.2.1 explains Verdict's architecture and overall workflow. Section 3.2.2 presents the class of SQL queries currently supported by Verdict. Section 3.2.3 introduces Verdict's query representation. Section 3.2.4 describes the intuition behind Verdict's inference. Lastly, section 3.2.5 discusses the limitations of Verdict's approach.

3.2.1 Architecture and Workflow

Verdict consists of a *query synopsis*, a *model*, and three processing modules: an *inference module*, a *learning module*, and an off-the-shelf *AQP engine*. Figure 3.2 depicts the connection between these components.

Term	Definition
raw answer	answer computed by the AQP engine
raw error	expected error for raw answer
improved answer	answer updated by Verdict
improved error	expected error for improved answer (by Verdict)
past snippet	supported query snippet processed in the past
new snippet	incoming query snippeet

Table 3.1: Terminology.

We begin by defining *query snippets*, which serve as the basic units of inference in Verdict.

Definition 1. (Query Snippet) A query snippet is a *supported* SQL query whose answer is a *single scalar value*, where supported queries are formally defined in section 3.2.2.

Section 3.2.3 describes how a supported query (whose answer may be a set) is decomposed into possibly multiple query snippets. For simplicity, and without loss of generality, here we assume that every incoming query is a query snippet.

For the i-th query snippet q_i, the AQP engine's answer includes a pair of an approximate answer $\tilde{\theta}_i$ and a corresponding expected error β_i. $\tilde{\theta}_i$ and β_i are formally defined in section 3.3.1, and are produced by most AQP systems [15,68,134,184,185,187]. Now we can formally define the first key component of our system, the *query synopsis*.

Definition 2. (Query Synopsis) Let n be the number of query snippets processed thus far by the AQP engine. The query synopsis Q_n is defined as the following set: $\{(q_i, \tilde{\theta}_i, \beta_i) \mid i = 1, \ldots, n\}$.

We call the query snippets in the query synopsis *past snippets*, and call the $(n+1)$-th query snippet the *new snippet*.

The second key component is the *model*, which represents Verdict's statistical understanding of the underlying data. The model is trained on the query synopsis, and is updated every time a query is added to the synopsis (section 3.4).

The query-time workflow of Verdict is as follows. Given an incoming query snippet q_{n+1}, Verdict invokes the AQP engine to compute a raw answer $\tilde{\theta}_{n+1}$ and a raw error β_{n+1}. Then, Verdict combines this raw answer/error and the previously computed model to *infer* an *improved answer* $\hat{\theta}_{n+1}$ and an associated expected error $\hat{\beta}_{n+1}$, called *improved error*. Theorem 3.1 shows that the improved error is never larger than the raw error. After returning the improved answer and the improved error to the user, $(q_{n+1}, \tilde{\theta}_{n+1}, \beta_{n+1})$ is added to the query synopsis.

A key objective in Verdict's design is to treat the underlying AQP engine as a black box. This allows Verdict to be used with many of the existing engines without requiring any modifications. From the user's perspective, the benefit of using Verdict (compared to using the AQP engine alone) is the error reduction and speedup, or only the error reduction, depending on the type of AQP engine used (section 3.6).

Lastly, Verdict does not modify non-aggregate expressions or unsupported queries, i.e., it simply returns their raw answers/errors to the user. Table 3.1 summarizes the terminology defined above. In section 3.3, we will recap the mathematical notations defined above.

3.2.2 Supported Queries

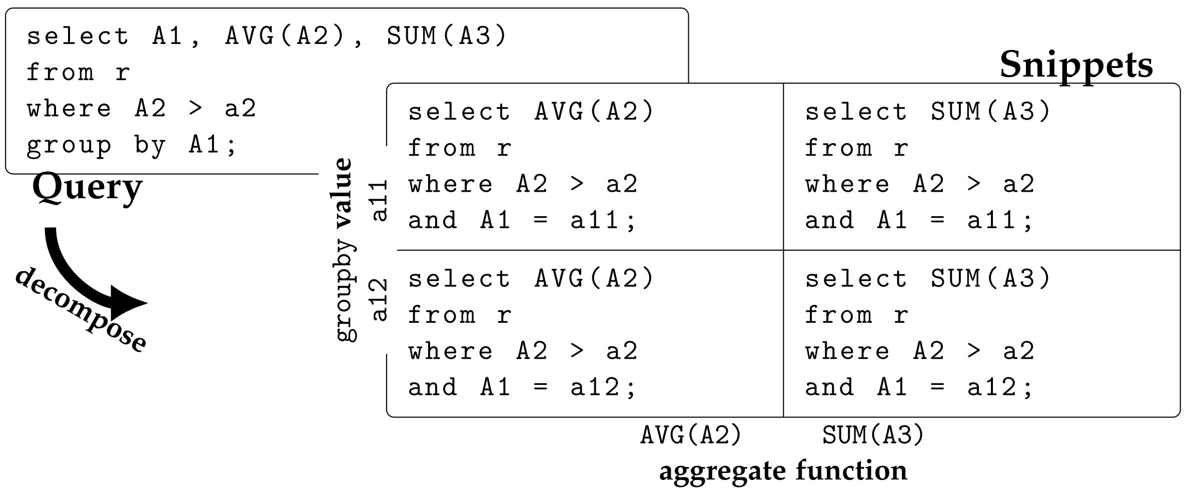
Verdict supports aggregate queries that are flat (i.e., no derived tables or sub-queries) with the following conditions:

1. **Aggregates**. Any number of `SUM`, `COUNT`, or `AVG` aggregates can appear in the `select` clause. The arguments to these aggregates can also be a *derived attribute*.
2. **Joins**. Verdict supports foreign-key joins between a fact table[2] and any number of dimension tables, exploiting the fact that this type of join does not introduce a sampling bias [11]. For simplicity, our discussion in this chapter is based on a denormalized table.
3. **Selections**. Verdict currently supports equality and inequality comparisons for categorical and numeric attributes (including the `in` operator). Currently, Verdict does not support disjunctions and textual filters (e.g., `like '%Apple%'`) in the `where` clause.
4. **Grouping**. `groupby` clauses are supported for both stored and derived attributes. The query may also include a `having` clause. Note that the underlying AQP engine may affect the cardinality of the result set depending on the `having` clause (e.g., subset/-superset error). Verdict simply operates on the result set returned by the AQP engine.

Nested Query Support— Although Verdict does not directly support nested queries, many queries can be flattened using joins [1] or by creating intermediate views for sub-queries [65]. In fact, this is the process used by Hive for supporting the nested queries of the TPC-H benchmark [82]. We are currently working to automatically process nested queries and to expand the class of supported queries (see section 5.8).

Unsupported Queries— Each query, upon its arrival, is inspected by Verdict's query type checker to determine whether it is supported, and if not, Verdict bypasses the

[2]Data warehouses typically record measurements (e.g., sales) into fact tables and normalize commonly appearing attributes (e.g., seller information) into dimension tables [161].



```
select A1, AVG(A2), SUM(A3)
from r
where A2 > a2
group by A1;
```

```
select AVG(A2)
from r
where A2 > a2
and A1 = a11;
```

```
select SUM(A3)
from r
where A2 > a2
and A1 = a11;
```

```
select AVG(A2)
from r
where A2 > a2
and A1 = a12;
```

```
select SUM(A3)
from r
where A2 > a2
and A1 = a12;
```

Figure 3.3: Example of a query's decomposition into multiple snippets.

INFERENCE module and simply returns the raw answer to the user. The overhead of the query type checker is negligible (section 3.11.5) compared to the runtime of the AQP engine; thus, Verdict does not incur any noticeable runtime overhead, even when a query is not supported.

Only supported queries are stored in Verdict's query synopsis and used to improve the accuracy of answers to future supported queries. That is, the class of queries that can be improved is equivalent to the class of queries that can be used to improve other queries.

3.2.3 Internal Representation

Decomposing Queries into Snippets— As mentioned in section 3.2.1, each supported query is broken into (possibly) multiple *query snippets* before being added to the query synopsis. Conceptually, each snippet corresponds to a supported SQL query with a single aggregate function, with no other projected columns in its `select` clause, and with no `groupby` clause; thus, the answer to each snippet is a single scalar value. A SQL query with multiple aggregate functions (e.g., `AVG(A2)`, `SUM(A3)`) or a `groupby` clause is converted to a set of multiple snippets for all combinations of each aggregate function and each `groupby` column value. As shown in the example of Figure 3.3, each `groupby` column value is added as an equality predicate in the `where` clause. The number of generated snippets can be extremely large, e.g., if a `groupby` clause includes a primary key. To ensure that the number of snippets added per each query is bounded, Verdict only generates snippets for N^{max} (1,000 by default) groups in the answer set. Verdict computes improved answers only for those snippets in order to bound the computational overhead.[3] For each aggregate function g, the query synopsis retains a maximum of C_g query snippets by following a least recently used snippet replacement policy (by default, C_g=2,000). This improves the efficiency of the inference process, while maintaining an accurate model based on the recently processed snippet answers.

Aggregate Computation— Verdict uses two aggregate functions to perform its internal computations: `AVG(`A_k`)` and `FREQ(*)`. As stated earlier, the attribute A_k can be either a stored attribute (e.g., `revenue`) or a derived one (e.g., `revenue * discount`). At runtime, Verdict combines these two types of aggregates to compute its supported aggregate functions as follows:

- `AVG(`A_k`)` = `AVG(`A_k`)`
- `COUNT(*)` = round(`FREQ(*)` $\times$ (table cardinality))
- `SUM(`A_k`)` = `AVG(`A_k`)` $\times$ `COUNT(*)`

3.2.4 Why and When Verdict Offers Benefit

In this section, we provide the high level intuition behind Verdict's approach to improving the quality of new snippet answers. Verdict exploits potential correlations between snippet answers to infer the answer of a new snippet. Let S_i and S_j be multisets of attribute values such that, when aggregated, they output exact answers to queries q_i and q_j, respectively. Then, the answers to q_i and q_j are correlated, if:

1. **S_i and S_j include common values.** $S_i \cap S_j \neq \phi$ implies the existence of correlation between the two snippet answers. For instance, the average household income of the state of Texas and the average household income of the United States are correlated, since these averages include some common values when computing those averages (here, the household incomes of Texas). In the TPC-H benchmark, 12 out of the 14 supported queries share common values in their aggregations.

2. **S_i and S_j include correlated values.** For instance, the average prices of a stock over two consecutive days are likely to be similar even if they do not share common values. When the compared days are farther apart, the similarity in their average stock prices might be lower. Verdict captures the likelihood of such attribute value similarities using a statistical measure called *inter-tuple covariance*, which will be formally defined in section 3.4.2. In the presence of non-zero inter-tuple covariances, the answers to q_i and q_j could be correlated even when $S_i \cap S_j \neq \phi$. In practice, most real-life datasets tend

[3]Dynamically adjusting the value of N^{max} (e.g., based on available resources and workload characteristics) makes an interesting direction for future work.

to have non-zero inter-tuple covariances, i.e., correlated attribute values (see section 3.12 for an empirical study). Note that this case includes the first case (in which S_i and S_j include common values), since the common values are always correlated in themselves.

Verdict formally captures the correlations between pairs of snippets using a probabilistic distribution function. At query time, this probabilistic distribution function is used to infer the most likely answer to the new snippet given the answers to past snippets.

3.2.5 Limitations

Verdict's model is the most likely explanation of the underlying distribution given the limited information stored in the query synopsis. Consequently, when a new snippet involves tuples that have never been accessed by past snippets, it is possible that Verdict's model might incorrectly represent the underlying distribution, and return incorrect error bounds. To guard against this limitation, Verdict always validates its model-based answer against the (model-free) answer of the AQP engine. We present this model validation step in section 3.9.

Because Verdict relies on off-the-shelf AQP engines for obtaining raw answers and raw errors, it is naturally bound by the limitations of the underlying engine. For example, it is known that sample-based engines are not apt at supporting arbitrary joins or `MIN`/`MAX` aggregates. Similarly, the validity of Verdict's error guarantees are contingent upon the validity of the AQP engine's raw errors. Fortunately, there are also off-the-shelf diagnostic techniques to verify the validity of such errors [14].

3.3 Inference

In this section, we describe Verdict's inference process for computing an improved answer (and improved error) for the new snippet. Verdict's inference process follows the standard machine learning arguments: we can understand in part the true distribution by means of observations, then we apply our understanding to predicting the unobserved. To this end, Verdict applies well-established techniques, such as the principle of maximum entropy and kernel-based estimations, to an AQP setting.

To present our approach, we first formally state our problem in section 3.3.1. A mathematical interpretation of the problem and the overview on Verdict's approach is described in section 3.3.2. Sections 3.3.3 and 3.3.4 present the details of the Verdict's

Sym.	Meaning
q_i	i-th (supported) query snippet
$n+1$	index number for a new snippet
$\boldsymbol{\theta}_i$	random variable representing our knowledge of the raw answer to q_i
$\tilde{\theta}_i$	(actual) raw answer computed by AQP engine for q_i
β_i	expected error associated with $\tilde{\theta}_i$
$\bar{\boldsymbol{\theta}}_i$	random variable representing our knowledge of the *exact* answer to q_i
$\bar{\theta}_i$	exact answer to q_i
$\hat{\theta}_{n+1}$	improved answer to the new snippet
$\hat{\beta}_{n+1}$	improved error to the new snippet

Table 3.2: Mathematical Notations.

approach to solving the problem. Section 3.3.5 discusses some challenges in applying Verdict's approach.

3.3.1 Problem Statement

Let r be a relation drawn from some unknown underlying distribution. r can be a join or Cartesian product of multiple tables. Let r's attributes be $A_1, \ldots, A_m$, where $A_1, \ldots, A_l$ are the *dimension attributes* and $A_{l+1}, \ldots, A_m$ are the *measure attributes*. Dimension attributes cannot appear inside aggregate functions while measure attributes can. Dimension attributes can be numeric or categorical, but measure attributes are numeric. Measure attributes can also be *derived attributes*. Table 3.2 summarizes the notations we defined earlier in section 3.2.1.

Given a query snippet q_i on r, an AQP engine returns a raw answer $\tilde{\theta}_i$ along with an associated expected error β_i. Formally, β_i^2 is the expectation of the squared deviation of $\tilde{\theta}_i$ from the (unknown) exact answer $\bar{\theta}_i$ to q_i.[4] β_i and β_j are independent if $i \neq j$.

Suppose n query snippets have been processed, and therefore the query synopsis Q_n contains the raw answers and raw errors for the past n query snippets. Without loss of generality, we assume all queries have the same aggregate function g on A_k (e.g., `AVG`(A_k)), where A_k is one of the measure attributes. Our problem is then stated as follows: *given Q_n and ($\tilde{\theta}_{n+1}$, β_{n+1}), compute the most likely answer to q_{n+1} with an associated expected error.*

In our discussion, for simplicity, we assume static data, i.e., the new snippet is issued against the same data that has been used for answering past snippets in Q_n. However, Verdict can also be extended to situations where the relations are subject to new data

[4]Here, the expectation is made over $\tilde{\theta}_i$ since the value of $\tilde{\theta}_i$ depends on samples.

being added, i.e., each snippet is answered against a potentially different version of the dataset. The generalization of Verdict under data updates is presented in section 3.10.

3.3.2 Inference Overview

In this section, we present our random variable interpretation of query answers and a high-level overview of Verdict's inference process.

Our approach uses (probabilistic) random variables to represent *our knowledge of the query answers.* The use of random variables here is a natural choice as our knowledge itself of the query answers is uncertain. Using random variables to represent degrees of belief is a standard approach in Beyesian inference. Specifically, we denote our knowledge of the raw answer and the exact answer to the i-th query snippet by random variables $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_i$, respectively. At this step, the only information available to us regarding $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_i$ is that θ_i is an instance of $\boldsymbol{\theta}_i$; no other assumptions are made.

Next, we represent the relationship between the set of random variables $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$, $\bar{\boldsymbol{\theta}}_{n+1}$ using a joint probability distribution function (pdf). Note that the first $n+1$ random variables are for the raw answers to past n snippets and the new snippet, and the last random variable is for the exact answer to the new snippet. We are interested in the relationship among those random variables because our knowledge of the query answers is based on limited information: the raw answers computed by the AQP engine, whereas we aim to find the most likely value for the new snippet's exact answer. This joint pdf represents Verdict's *prior belief* over the query answers. We denote the joint pdf by $f(\boldsymbol{\theta}_1 = \theta'_1, \ldots, \boldsymbol{\theta}_{n+1} = \theta'_{n+1}, \bar{\boldsymbol{\theta}}_{n+1} = \bar{\theta}'_{n+1})$. For brevity, we also use $f(\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1})$ when the meaning is clear from the context. (Recall that θ_i refers to an actual raw answer from the AQP engine, and $\bar{\theta}_{n+1}$ refers to the exact answer to the new snippet.) The joint pdf returns the probability that the random variables $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$ takes a particular combination of the values, i.e., $\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1}$. In Section 3.3.3, we discuss how to obtain this joint pdf from some statistics available on query answers.

Then, we compute the most likely value for the new snippet's exact answer, namely the most likely value for $\bar{\boldsymbol{\theta}}_{n+1}$, by first conditionalizing the joint pdf on the actual observations (i.e., raw answers) from the AQP engine, i.e., $f(\bar{\boldsymbol{\theta}}_{n+1} = \bar{\theta}'_{n+1} \mid \boldsymbol{\theta}_1 = \theta_1, \ldots, \boldsymbol{\theta}_{n+1} = \theta_{n+1})$. We then find the value of $\bar{\theta}'_{n+1}$ that maximizes the conditional pdf. We call this value the *model-based answer* and denote it by $\ddot{\theta}_{n+1}$. Section 3.3.4 provides more details of this process. Finally, $\ddot{\theta}_{n+1}$ and its associated expected error $\ddot{\beta}_{n+1}$ are returned as Verdict's improved answer and improved error if they pass the model validation (described in section 3.9). Otherwise, the (original) raw answer and error are taken as Verdict's improved answer and error, respectively. In other words, if the model validation fails, Verdict simply returns the original raw results from the AQP engine without any improvements.

3.3.3 Prior Belief

In this section, we describe how Verdict obtains a joint pdf $f(\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1})$ that represents its knowledge of the underlying distribution. The intuition behind Verdict's inference is to make use of possible correlations between pairs of query answers. This section applies such statistical information of query answers (i.e., means, covariances, and variances) for obtaining the most likely joint pdf. Obtaining the query statistics is described in section 3.4.

To obtain the joint pdf, Verdict relies on the principle of maximum entropy (ME) [31, 162], a simple but powerful statistical tool for determining a pdf of random variables given some statistical information available. According to the ME principle, given some testable information on random variables associated with a pdf in question, the pdf that best represents the current state of our knowledge is the one that maximizes the following expression, called *entropy*:

$$h(f) = -\int f(\vec{\theta}) \cdot \log f(\vec{\theta}) \, d\vec{\theta} \tag{3.1}$$

where $\vec{\theta} = (\theta'_1, \ldots, \theta'_{n+1}, \bar{\theta}'_{n+1})$.

Note that the joint pdf maximizing the above entropy differs depending on the kinds of given testable information, i.e., query statistics in our context. For instance, the maximum entropy pdf given means of random variables is different from the maximum entropy pdf given means and (co)variances of random variables. In fact, there are two conflicting considerations when applying this principle. On one hand, the resulting pdf can be computed more efficiently if the provided statistics are simple or few, i.e., simple statistics reduce the computational complexity. On the other hand, the resulting pdf can describe the relationship among the random variables more accurately if richer statistics are provided, i.e., the richer the statistics, the more accurate our improved answers. Therefore, we need to choose an appropriate degree of statistical information to strike a balance between the computational efficiency of pdf evaluation and its accuracy in describing the relationship among query answers.

Among possible options, Verdict uses the first and the second order statistics of the random variables, i.e., mean, variances, and covariances. The use of second-order statistics enables us to capture the relationship among the answers to different query snippets, while the joint pdf that maximizes the entropy can be expressed in an analytic form. The uses of analytic forms provides computational efficiency. Specifically, the joint pdf that maximizes the entropy while satisfying the given means, variances, and covariances is a multivariate normal with the corresponding means, variances, and covariances [162].

Lemma 1. Let $\vec{\boldsymbol{\theta}} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})^\intercal$ be a vector of $n{+}2$ random variables with mean values $\vec{\mu} = (\mu_1, \ldots, \mu_{n+1}, \bar{\mu}_{n+1})^\intercal$ and a $(n{+}2) \times (n{+}2)$ covariance matrix Σ specifying their variances and pairwise covariances. The joint pdf f over these random variables that maximizes $h(f)$ while satisfying the provided means, variances, and covariances is the following function:

$$f(\vec{\theta}) = \frac{1}{\sqrt{(2\pi)^{n+2}|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{\theta} - \vec{\mu})^\intercal \Sigma^{-1} (\vec{\theta} - \vec{\mu})\right), \tag{3.2}$$

and this solution is unique.

In the following section, we describe how Verdict computes the most likely answer to the new snippet using this joint pdf in Equation (3.2). We call the most likely answer a *model-based answer*. In section 3.9, this model-based answer is chosen as an improved answer if it passes a model validation. Finally, in section 3.3.5, we discuss the challenges involved in obtaining $\vec{\mu}$ and Σ, i.e., the query statistics required for deriving the joint pdf.

3.3.4 Model-based Answer

In the previous section, we formalized the relationship among query answers, namely $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$, using a joint pdf. In this section, we exploit this joint pdf to infer the most likely answer to the new snippet. In other words, we aim to find the most likely value for $\bar{\boldsymbol{\theta}}_{n+1}$ (the random variable representing q_{n+1}'s exact answer), given the observed values for $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$, i.e., the raw answers from the AQP engine. We call the most likely value a *model-based answer* and its associated expected error a *model-based error*. Mathematically, Verdict's model-based answer $\ddot{\theta}_{n+1}$ to q_{n+1} can be expressed as:

$$\ddot{\theta}_{n+1} = \underset{\bar{\theta}'_{n+1}}{\arg\max} f(\bar{\theta}'_{n+1} \mid \boldsymbol{\theta}_1 = \tilde{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1} = \tilde{\theta}_{n+1}). \tag{3.3}$$

That is, $\ddot{\theta}_{n+1}$ is the value at which the conditional pdf has its maximum value. The conditional pdf, $f(\bar{\theta}'_{n+1} \mid \theta_1, \ldots, \theta_{n+1})$, is obtained by conditioning the joint pdf obtained in section 3.3.3 on the observed values, i.e., raw answers to the past snippets and the new snippet.

Computing a conditional pdf may be a computationally expensive task. However, a conditional pdf of a multivariate normal distribution is analytically computable; it is another normal distribution. Specifically, the conditional pdf in Equation (3.3) is a normal distribution with the following mean μ_c and variance σ_c^2 [32]:

$$\mu_c = \bar{\mu}_{n+1} + k_{n+1}^\intercal \Sigma_{n+1}^{-1} (\vec{\theta}_{n+1} - \vec{\mu}_{n+1}) \tag{3.4}$$

$$\sigma_c^2 = \bar{\kappa}^2 - k_{n+1}^\intercal \Sigma_{n+1}^{-1} k_{n+1} \tag{3.5}$$

where:

- k_{n+1} is a column vector of length $n+1$ whose i-th element is $(i, n+2)$-th entry of Σ;
- Σ_{n+1} is a $(n+1) \times (n+1)$ submatrix of Σ consisting of Σ's first $n+1$ rows and columns;
- $\vec{\theta}_{n+1}{=}(\tilde{\theta}_1, \ldots, \tilde{\theta}_{n+1})^\intercal$;
- $\vec{\mu}_{n+1} = (\mu_1, \ldots, \mu_{n+1})^\intercal$; and
- $\bar{\kappa}^2$ is the $(n+2, n+2)$-th entry of Σ

Since the mean of a normal distribution is the value at which the pdf takes a maximum value, we take μ_c as our model-based answer $\ddot{\theta}_{n+1}$. Likewise, the expectation of the squared deviation of the value $\bar{\theta}'_{n+1}$, which is distributed according to the conditional pdf in Equation (3.3), from the model-based answer $\ddot{\theta}_{n+1}$ coincides with the variance σ_c^2 of the conditional pdf. Thus, we take σ_c as our model-based error $\ddot{\beta}_{n+1}$.

Computing each of Equations (3.4) and (3.5) requires $O(n^3)$ time complexity at query time. However, Verdict uses alternative forms of these equations that require only $O(n^2)$ time complexity at query time (section 3.7). As a future work, we plan to employ inferential techniques with sub-linear time complexity [102, 176] for a more sophisticated eviction policy for past queries.

Note that, since the conditional pdf is a normal distribution, the error bound at confidence δ is expressed as $\alpha_\delta \cdot \ddot{\beta}_{n+1}$, where α_δ is a non-negative number such that a random number drawn from a standard normal distribution would fall within $(-\alpha_\delta, \alpha_\delta)$ with probability δ. We call α_δ the confidence interval multiplier for probability δ. That is, the exact answer $\bar{\theta}_{n+1}$ is within the range $(\ddot{\theta}_{n+1} - \alpha_\delta \cdot \ddot{\beta}_{n+1},\ \ddot{\theta}_{n+1} + \alpha_\delta \cdot \ddot{\beta}_{n+1})$ with probability δ, according to Verdict's model.

3.3.5 Key Challenges

As mentioned in section 3.3.3, obtaining the joint pdf in Lemma 1 (which represents Verdict's prior belief on query answers) requires the knowledge of means, variances, and covariances of the random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$. However, acquiring these statistics is a non-trivial task for two reasons. First, we have only observed one value for each of the random values $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}$, namely $\tilde{\theta}_1, \ldots, \tilde{\theta}_{n+1}$. Estimating variances and covariances of random variables from a single value is nearly impossible. Second, we do not have any observation for the last random variable $\bar{\boldsymbol{\theta}}_{n+1}$ (recall that $\bar{\boldsymbol{\theta}}_{n+1}$ represents our knowledge of the exact answer to the new snippet, i.e., $\bar{\theta}_{n+1}$). In section 3.4, we present Verdict's approach to solving these challenges.

3.4 Estimating Query Statistics

As described in section 3.3, Verdict expresses its prior belief on the relationship among query answers as a joint pdf over a set of random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$. In this process, we need to know the means, variances, and covariances of these random variables.

Verdict uses the arithmetic mean of the past query answers for the mean of each random variable, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$. Note that this only serves as a prior belief, and will be updated in the process of conditioning the prior belief using the observed query answers. In this section, without loss of generality, we assume the mean of the past query answers is zero.

Thus, in the rest of this section, we focus on obtaining the variances and covariances of these random variables, which are the elements of the $(n+2) \times (n+2)$ covariance matrix Σ in lemma 1 (thus, we can obtain the elements of the column vector k_{n+1} and the variance $\bar{\kappa}^2$ as well). Note that, due to the independence between expected errors, we have:

$$\begin{aligned} \mathrm{cov}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) &= \mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) + \delta(i,j) \cdot \beta_i^2 \\ \mathrm{cov}(\boldsymbol{\theta}_i, \bar{\boldsymbol{\theta}}_j) &= \mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) \end{aligned} \tag{3.6}$$

where $\delta(i,j)$ returns 1 if $i = j$ and 0 otherwise. Thus, computing $\mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ is sufficient for obtaining Σ.

Computing $\mathrm{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ relies on a straightforward observation: *the covariance between two query snippet answers is computable using the covariances between the attribute values involved in computing those answers.* For instance, we can easily compute the covariance between (i) the average revenue of the years 2014 and 2015 and (ii) the average revenue of the years 2015 and 2016, as long as we know the covariance between the average revenues of every pair of days in 2014–2016.

In this work, we further extend the above observation. That is, if we are able to compute the covariance between the average revenues at an infinitesimal time t and another infinitesimal time t', we will be able to compute the covariance between (i) the average revenue of 2014–2015 and (ii) the average revenue of 2015–2016, by integrating the covariances between the revenues at infinitesimal times over appropriate ranges. Here, the covariance between the average revenues at two infinitesimal times t and t' is defined in terms of the underlying data distribution that has generated the relation r, where the past query answers help us discover the most-likely underlying distribution. The rest of this section formalizes this idea.

In section 3.4.1, we present a decomposition of the (co)variances between pairs of query snippet answers into *inter-tuple covariance* terms. Then, in section 3.4.2, we describe how inter-tuple covariances can be estimated analytically using parameterized functions.

3.4.1 Covariance Decomposition

To compute the variances and covariances between query snippet answers (i.e., $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1}$), Verdict relies on our proposed *inter-tuple covariances*, which express the statistical properties of the underlying distribution. Before presenting the inter-tuple covariances, our discussion starts with the fact that the answer to a supported snippet can be mathematically represented in terms of the underlying distribution. This representation then naturally leads us to the decomposition of the covariance between query answers into smaller units, which we call inter-tuple covariances.

Let g be an aggregate function on attribute A_k, and $\boldsymbol{t} = (a_1, \ldots, a_l)$ be a vector of length l comprised of the values for r's dimension attributes $A_1, \ldots, A_l$. To help simplify the mathematical descriptions in this section, we assume that all dimension attributes are numeric (not categorical), and the selection predicates in queries may contain range constraints on some of those dimension attributes. Handling categorical attributes is a straightforward extension of this process (see section 3.13.2).

We define a continuous function $\nu_g(\boldsymbol{t})$ for every aggregate function g (e.g., `AVG(`A_k`)`, `FREQ(*)`) such that, when integrated, it produces answers to query snippets. That is (omitting possible normalization and weight terms for simplicity):

$$\bar{\boldsymbol{\theta}}_i = \int_{\boldsymbol{t} \in F_i} \nu_g(\boldsymbol{t}) \, d\boldsymbol{t} \tag{3.7}$$

Formally, F_i is a subset of the Cartesian product of the domains of the dimension attributes, $A_1, \ldots, A_l$, such that $\boldsymbol{t} \in F_i$ satisfies the selection predicates of q_i. Let $(s_{i,k}, e_{i,k})$ be the range constraint for A_k specified in q_i. We set the range to $(\min(A_k), \max(A_k))$

if no constraint is specified for A_k. Verdict simply represents F_i as the product of those l per-attribute ranges. Thus, the above Equation (3.7) can be expanded as:

$$\bar{\boldsymbol{\theta}}_i = \int_{s_{i,l}}^{e_{i,l}} \cdots \int_{s_{i,1}}^{e_{i,1}} \nu_g(\boldsymbol{t}) \, da_1 \cdots da_l$$

For brevity, we use the single integral representation using F_i unless the explicit expression is needed.

Using Equation (3.7) and the linearity of covariance, we can decompose $\text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ into:

$$\begin{aligned} \text{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) &= \text{cov}\left(\int_{\boldsymbol{t} \in F_i} \nu_g(\boldsymbol{t}) \, d\boldsymbol{t}, \int_{\boldsymbol{t}' \in F_j} \nu_g(\boldsymbol{t}') \, d\boldsymbol{t}' \right) \\ &= \int_{\boldsymbol{t} \in F_i} \int_{\boldsymbol{t}' \in F_j} \text{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}')) \, d\boldsymbol{t} \, d\boldsymbol{t}' \end{aligned} \tag{3.8}$$

As a result, the covariance between query answers can be broken into an integration of the covariances between tuple-level function values, which we call *inter-tuple covariances*.

To use Equation (3.8), we must be able to compute the inter-tuple covariance terms. However, computing these inter-tuple covariances is challenging, as we only have a single observation for each $\nu_g(\boldsymbol{t})$. Moreover, even if we had a way to compute the inter-tuple covariance for arbitrary $\boldsymbol{t}$ and $\boldsymbol{t}'$, the exact computation of Equation (3.8) would still require an infinite number of inter-tuple covariance computations, which would be infeasible. In the next section, we present an efficient alternative for estimating these inter-tuple covariances.

3.4.2 Analytic Inter-tuple Covariances

To efficiently estimate the inter-tuple covariances, and thereby compute Equation (3.8), we propose using *analytical covariance functions*, a well-known technique in statistical literature for approximating covariances [32]. In particular, Verdict uses squared exponential covariance functions, which is capable of approximating any continuous target function arbitrarily closely as the number of observations (here, query answers) increases [122].[5] Although the underlying distribution may not be a continuous function, it is sufficient for us to obtain $\nu_g(\boldsymbol{t})$ such that, when integrated (as in Equation (3.7)), produces

[5]This property of the universal kernels is asymptotic (i.e., as the number of observations goes to infinity).

the same values as the integrations of the underlying distribution.[6] In our setting, the squared exponential covariance function $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ is defined as:

$$\operatorname{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}')) \approx \rho_g(\boldsymbol{t}, \boldsymbol{t}') = \sigma_g^2 \cdot \prod_{k=1}^{l} \exp\left(-\frac{(a_k - a_k')^2}{l_{g,k}^2} \right) \tag{3.9}$$

Here, $l_{g,k}$ for $k{=}1 \ldots l$ and σ_g^2 are tunable *correlation parameters* to be learned from past queries and their answers (section 3.8).

Intuitively, when $\boldsymbol{t}$ and $\boldsymbol{t}'$ are similar, i.e., $(a_k - a_k')^2$ is small for most A_k, then $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ returns a larger value (closer to σ_g^2), indicating that the expected values of g for $\boldsymbol{t}$ and $\boldsymbol{t}'$ are highly correlated.

With the analytic covariance function above, the $\operatorname{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ terms involving inter-tuple covariances can in turn be computed analytically. Note that Equation (3.9) involves the multiplication of l terms, each of which containing variables related to a single attribute. As a result, plugging Equation (3.9) into Equation (3.8) yields:

$$\operatorname{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j) = \sigma_g^2 \prod_{k=1}^{l} \int_{s_{i,k}}^{e_{i,k}} \int_{s_{j,k}}^{e_{j,k}} \exp\left(-\frac{(a_k - a_k')^2}{l_{g,k}^2} \right) da_k' a_k \tag{3.10}$$

The order of integrals are interchangeable, since the terms including no integration variables can be regarded as constants (and thus can be factored out of the integrals). Note that the double-integral of an exponential function can also be computed analytically (see section 3.13.1); thus, Verdict can efficiently compute $\operatorname{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ in $O(l)$ times by directly computing the integrals of inter-tuple covariances, without explicitly computing individual inter-tuple covariances. Finally, we can compose the $(n+2) \times (n+2)$ matrix Σ in Lemma 1 using Equation (3.6).

3.5 Verdict Process Summary

In this section, we summarize Verdict's offline and online processes. Suppose the query synopsis Q_n contains a total of n query snippets from past query processing, and a new query is decomposed into b query snippets; we denote the new query snippets in the new query by $q_{n+1}, \ldots, q_{n+b}$.

[6]The existence of such a continuous function is implied by the kernel density estimation technique [172].

Algorithm 1: Verdict offline process

```
Input: $Q_n$ including $(q_i, \theta_i, \beta_i)$ for $i = 1, \ldots, n$
Output: $Q_n$ with new model parameters and precomputed matrices
1 foreach aggregate function $g$ in $Q_n$ do
2     $(l_{g,1}, \ldots, l_{g,l}, \sigma_g^2) \leftarrow$ learn($Q_n$)        // section 3.8
      // $\Sigma_{(i,j)}$ indicates $(i,j)$-element of $\Sigma$
3     for $(i,j) \leftarrow (1,\ldots,n) \times (1,\ldots,n)$ do
          // Equation (3.6)
4         $\Sigma_{(i,j)} \leftarrow$ covariance($q_i, q_j; l_{g,1}, \ldots, l_{g,l}, \sigma_g^2$)
5     end
6     Insert $\Sigma$ and $\Sigma^{-1}$ into $Q_n$ for $g$
7 end
8 return $Q_n$
```

Offline processing— Algorithm 1 summarizes Verdict's offline process. It consists of learning correlation parameters and computing covariances between all pairs of past query snippets.

Online processing— Algorithm 2 summarizes Verdict's runtime process. Here, we assume the new query is a supported query; otherwise, Verdict simply forwards the AQP engine's query answer to the user.

3.6 Deployment Scenarios

Verdict is designed to support a large class of AQP engines. However, depending on the type of AQP engine used, Verdict may provide both speedup and error reduction, or only error reduction.

1. **AQP engines that support online aggregation** [68,134,184,185]: Online aggregation continuously refines its approximate answer as new tuples are processed, until users are satisfied with the current accuracy or when the entire dataset is processed. In these types of engines, every time the online aggregation provides an updated answer (and error estimate), Verdict generates an improved answer with a higher accuracy (by paying small runtime overhead). As soon as this accuracy meets the user requirement, the online aggregation can be stopped. With Verdict, the online aggregation's continuous processing will stop earlier than it would without Verdict. This is because Verdict reaches a target error bound much earlier by combining its model with the raw answer of the AQP engine.

Algorithm 2: Verdict runtime process

```
Input: New query snippets $q_{n+1}, \ldots, q_{n+b}$,
       Query synopsis $Q_n$
Output: $b$ number of improved answers and improved errors
        $\{(\widehat{\theta}_{n+1}, \widehat{\beta}_{n+1}), \ldots, (\widehat{\theta}_{n+b}, \widehat{\beta}_{n+b})\}$,
        Updated query synopsis $Q_{n+b}$
1  fc ← number of distinct aggregate functions in new queries
   /* The new query (without decomposition) is sent to the AQP engine in
      practice.                                                          */
2  $\{(\theta_{n+1}, \beta_{n+1}), \ldots, (\theta_{n+b}, \beta_{n+b})\} \leftarrow$ AQP($q_{n+1}, \ldots, q_{n+b}$)
   // improve up to $N^{max}$ rows
3  for $i \leftarrow 1, \ldots, (\text{fc} \cdot N^{max})$ do
       // model-based answer/error
       // (Equations (3.4) and (3.5))
4      $(\ddot{\theta}_{n+i}, \ddot{\beta}_{n+i}) \leftarrow$ inference($\theta_{n+i}, \beta_{n+i}, Q_n$)
       // model validation (section 3.9)
5      $(\widehat{\theta}_{n+i}, \widehat{\beta}_{n+i}) \leftarrow$ if valid($\ddot{\theta}_{n+i}, \ddot{\beta}_{n+i}$) then $(\ddot{\theta}_{n+i}, \ddot{\beta}_{n+i})$ else $(\theta_{n+i}, \beta_{n+i})$
6      Insert $(q_{n+i}, \theta_{n+i}, \beta_{n+i})$ into $Q_n$
7  end
8  for $i \leftarrow (\text{fc} \cdot N^{max} + 1), \ldots, b$ do
9      $(\widehat{\theta}_{n+i}, \widehat{\beta}_{n+i}) \leftarrow (\theta_{n+i}, \beta_{n+i})$
10 end
   // Verdict overhead ends
11 return $\{(\widehat{\theta}_{n+1}, \widehat{\beta}_{n+1}), \ldots, (\widehat{\theta}_{n+b}, \widehat{\beta}_{n+b})\}, Q_n$
```

2. **AQP engines that support time-bounds** [15,38,47,71,86,144,187]: Instead of continuously refining approximate answers and reporting them to the user, these engines simply take a time-bound from the user, and then they predict the largest sample size that they can process within the requested time-bound; thus, they minimize error bounds within the allotted time. For these engines, Verdict simply replaces the user's original time bound t_1 with a slightly smaller value $t_1 - \epsilon$ before passing it down to the AQP engine, where ϵ is the time needed by Verdict for inferring the improved answer and improved error. Thanks to the efficiency of Verdict's inference, ϵ is typically a small value, e.g., a few milliseconds (see section 3.11.5). Since Verdict's inference brings larger accuracy improvements on average compared to the benefit of processing more tuples within the ϵ time, Verdict achieves significant error reductions over traditional AQP engines.

In this chapter, we use an online aggregation engine to demonstrate Verdict's both speedup and error reduction capabilities (section 5.6). However, for interested readers, we also provide evaluations on a time-bound engine section 3.11.11.

Some AQP engines also support error-bound queries but do not offer an online aggregation interface [16, 128, 146]. For these engine, Verdict currently only benefits their time-bound queries, leaving their answer to error-bound queries unchanged. Supporting the latter would require either adding an online aggregation interface to the AQP engine, or a tighter integration of Verdict and the AQP engine itself. Such modifications are beyond the scope of this chapter, as one of our design goals is to treat the underlying AQP engine as a black box (Figure 3.2), so that Verdict can be used alongside a larger number of existing engines.

Note that Verdict's inference mechanism is not affected by the specific AQP engine used underneath, as long as the conditions in section 3.3 hold, namely the error estimate β^2 is the expectation of the squared deviation of the approximate answer from the exact answer. However, the AQP engine's runtime overhead (e.g., query parsing and planning) may affect Verdict's overall benefit in relative terms. For example, if the query parsing amount to 90% of the overall query processing time, even if Verdict completely eliminates the need for processing any data, the relative speedup will only be $1.0/0.9 = 1.11\times$. However, Verdict is designed for data-intensive scenarios where disk or network I/O is a sizable portion of the overall query processing time.

3.7 Formal Guarantees

Next, we formally show that the error bounds of Verdict's improved answers are never larger than the error bounds of the AQP engine's raw answers.

Theorem 3.1. Let Verdict's improved answer and improved error to the new snippet be $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$ and the AQP engine's raw answer and raw error to the new snippet be $(\theta_{n+1}, \beta_{n+1})$. Then,

$$\hat{\beta}_{n+1} \leq \beta_{n+1}$$

and the equality occurs when the raw error is zero, or when Verdict's query synopsis is empty, or when Verdict's model-based answer is rejected by the model validation step.

Proof. Recall that $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$ is set either to Verdict's model-based answer/error, i.e., $(\ddot{\theta}_{n+1}, \ddot{\beta}_{n+1})$, or to the AQP system's raw answer/error, i.e., $(\theta_{n+1}, \beta_{n+1})$, depending on the result of the model validation. In the latter case, it is trivial that $\hat{\beta}_{n+1} \leq \beta_{n+1}$, and hence it is enough to show that $\ddot{\beta}_{n+1} \leq \beta_{n+1}$.

Computing $\ddot{\beta}_{n+1}$ involves an inversion of the covariance matrix Σ_{n+1}, where Σ_{n+1} includes the β_{n+1} term on one of its diagonal entries. We show $\ddot{\beta}_{n+1} \le \beta_{n+1}$ by directly simplifying $\ddot{\beta}_{n+1}$ into the form that involves β_{n+1} and other terms.

We first define notations. Let Σ be the covariance matrix of the vector of random variables $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n+1}, \bar{\boldsymbol{\theta}}_{n+1})$; k_n be a column vector of length n whose i-th element is the $(i, n+1)$-th entry of Σ; Σ_n be an $n \times n$ submatrix of Σ that consists of Σ's first n rows/columns; $\bar{\kappa}^2$ be a scalar value at the $(n+2, n+2)$-th entry of Σ; and $\vec{\theta}_n$ be a column vector $(\tilde{\theta}_1, \ldots, \tilde{\theta}_n)^\intercal$.

Then, we can express k_{n+1} and Σ_{n+1} in Equations (3.4) and (3.5) in block forms as follows:

$$k_{n+1} = \begin{pmatrix} k_n \\ \bar{\kappa}^2 \end{pmatrix}, \quad \Sigma_{n+1} = \begin{pmatrix} \Sigma_n & k_n \\ k_n^\intercal & \bar{\kappa}^2 + \beta_{n+1}^2 \end{pmatrix}, \quad \vec{\theta}_{n+1} = \begin{pmatrix} \vec{\theta}_n \\ \tilde{\theta}_{n+1} \end{pmatrix}$$

Here, it is important to note that k_{n+1} can be expressed in terms of k_n and $\bar{\kappa}^2$ because $(i, n+1)$-th element of Σ and $(i, n+2)$-th element of Σ have the same values for $i = 1, \ldots, n$. They have the same values because the covariance between $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{n+1}$ and the covariance between $\boldsymbol{\theta}_i$ and $\bar{\boldsymbol{\theta}}_{n+1}$ are same for $i = 1, \ldots, n$ due to Equation (3.6).

Using the formula of block matrix inversion [78], we can obtain the following alternative forms of Equations (3.4) and (3.5) (here, we assume zero means to simplify the expressions):

$$\gamma^2 = \bar{\kappa}^2 - k_n^\intercal \Sigma_n^{-1} k_n, \qquad \theta = k_n^\intercal \Sigma_n^{-1} \vec{\theta}_n \tag{3.11}$$

$$\ddot{\theta}_{n+1} = \frac{\beta_{n+1}^2 \cdot \theta + \gamma^2 \cdot \theta_{n+1}}{\beta_{n+1}^2 + \gamma^2}, \qquad \ddot{\beta}_{n+1}^2 = \frac{\beta_{n+1}^2 \cdot \gamma^2}{\beta_{n+1}^2 + \gamma^2} \tag{3.12}$$

Note that $\ddot{\beta}_{n+1}^2 < \beta_{n+1}$ for $\beta_{n+1} > 0$ and $\gamma^2 < \infty$, and $\ddot{\beta}_{n+1}^2 = \beta_{n+1}$ if $\beta_{n+1} = 0$ or $\gamma^2 \to \infty$. □

Lemma 2. The time complexity of Verdict's inference is $O(N^{max} \cdot l \cdot n^2)$ The space complexity of Verdict is $O(n \cdot N^{max} + n^2)$, where $n \cdot N^{max}$ is the size of the query snippets and n^2 is the size of the precomputed covariance matrix.

Proof. It is enough to prove that the computations of a model-based answer and a model-based error can be performed in $O(n^2)$ time, where n is the number of past query snippets. Note that this is clear from Equations (3.11) and (3.12), because the computation of Σ_n^{-1} involves only the past query snippets. For computing γ^2, multiplying k_n, a precomputed Σ_n^{-1}, and k_n takes $O(n^2)$ time. Similarly for θ in Equation (3.11) □

These results imply that the domain sizes of dimension attributes do not affect Verdict's computational overhead. This is because Verdict analytically computes the covariances between pairs of query answers without individually computing inter-tuple covariances (section 3.4.2).

3.8 Parameter Learning

In this section, we describe how to find the most likely values of the correlation parameters defined in section 3.4.2. In this process, we exploit the joint pdf defined in Equation (3.2), as it allows us to compute the likelihood of a certain combination of query answers given relevant statistics. Let $\vec{\theta}_{\text{past}}$ denote a vector of raw answers to past snippets. Then, by Bayes' theorem:

$$\Pr(\Sigma_n \mid \vec{\theta}_{\text{past}}) \propto \Pr(\Sigma_n) \cdot \Pr(\vec{\theta}_{\text{past}} \mid \Sigma_n)$$

where Σ_n is an $n \times n$ submatrix of Σ consisting of Σ's first n rows and columns, i.e., (co)variances between pairs of past query answers, and $\propto$ indicates that the two values are proportional, Therefore, without any preference over parameter values, determining the most likely correlation parameters (which determine Σ_n) given past queries amounts to finding the values for $l_{g,1}, \ldots, l_{g,l}, \sigma_g^2$ that maximize the below log-likelihood function:

$$\begin{aligned} \log \Pr(\vec{\theta}_{\text{past}} \mid \Sigma_n) &= \log f(\vec{\theta}_{\text{past}}) \\ &= -\frac{1}{2}\vec{\theta}_{\text{past}}^{\intercal}\Sigma_n^{-1}\vec{\theta}_{\text{past}} - \frac{1}{2}\log|\Sigma_n| - \frac{n}{2}\log 2\pi \end{aligned} \tag{3.13}$$

where $f(\vec{\theta}_{\text{past}})$ is the joint pdf from Equation (3.2).

Verdict finds the optimal values for $l_{g,1}, \ldots, l_{g,l}$ by solving the above optimization problem with a numerical solver, while it estimates the value for σ_g^2 analytically from past query answers (see section 3.13.3). Concretely, the current implementation of Verdict uses the gradient-descent-based (quasi-newton) nonlinear programming solver provided by Matlab's `fminuncon()` function, without providing explicit gradients. Although our current approach is typically slower than using closed-form solutions or than using the solver with an explicit gradient (and a Hessian), they do not pose a challenge in Verdict's setting, since all these parameters are computed *offline*, i.e., prior to the arrival of new queries. We plan to improve the efficiency of this offline training by using explicit gradient expressions.

Since Equation (3.13) is not a convex function, the solver of our choice only returns a locally optimal point. A conventional strategy to handle this issue is to obtain multiple locally optimal points by solving the same problem with multiple random starting points, and to take the one with the highest log-likelihood value as a final answer. Still, this approach does not guarantee the correctness of the model. In contrast, Verdict's strategy is to find a locally-optimal point that can capture potentially large inter-tuple covariances, and to validate the correctness of the resulting model against a model-free answer (section 3.9). We demonstrate empirically in the following section that this strategy is effective for finding parameter values that are close to true values. Verdict's model validation process in section 3.9 provides robustness against the models that may differ from the true distribution. Verdict uses $l_{g,k} = (\max(A_k) - \min(A_k))$ for the starting point of the optimization problem.

Lastly, our use of approximate answers as the constraints for the ME principle is properly accounted for by including additive error terms in their (co)variances (Equation (3.6)).

3.9 Model Validation

Verdict'd model validation rejects its model—the most likely explanation of the underlying distribution given the answers to past snippets—if there is evidence that its model-based error is likely to be incorrect. Verdict's model validation process addresses two situations: (i) negative estimates for `FREQ(*)`, and (ii) an unlikely large discrepancy between a model-based answer and a raw answer.

Negative estimates for `FREQ(*)`— To obtain the prior distribution of the random variables, Verdict uses the most-likely distribution (based on the maximum entropy principle (lemma 1)), given the means, variances, and covariances of query answers. Although this makes the inference analytically computable, the lack of explicit non-negative constraints on the query answers may produce negative estimates on `FREQ(*)`. Verdict handles this situation with a simple check; that is, Verdict rejects its model-based answer if $\ddot{\theta}_{n+1} < 0$ for `FREQ(*)`, and uses the raw answer instead. Even if $\ddot{\theta}_{n+1} \geq 0$, the lower bound of the confidence interval is set to zero if the value is less than zero.

Unlikely model-based answer— Verdict's model learned from empirical observations may be different from the true distribution. Figure 3.4(a) illustrates such an example. Here, after the first three queries, the model is consistent with past query answers (shown as gray boxes); however, it incorrectly estimates the distribution of the unob-

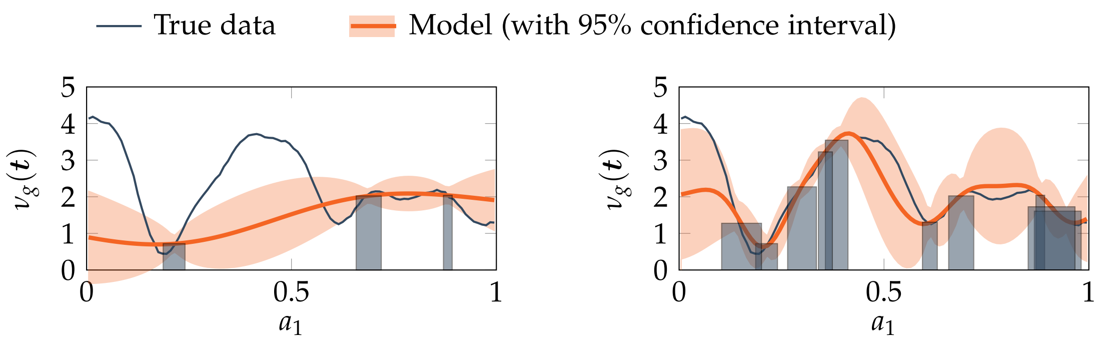


Figure 3.4: An example of (a) overly optimistic confidence intervals due to incorrect estimation of the underlying distributon, and (b) its resolution with more queries processed. Verdict relies on a model validation to avoid the situation as in (a).

served data, leading to overly optimistic confidence intervals. Figure 3.4(b) shows that the model becomes more consistent with the data as more queries are processed.

Verdict rejects (and does not use) its own model in situations such as Figure 3.4(a) by validating its model-based answers against the *model-free* answers obtained from the AQP engine. Specifically, we define a *likely region* as the range in which the AQP engine's answer would fall with high probability (99% by default) if Verdict's model were to be correct. If the AQP's raw answer θ_{n+1} falls outside this likely region, Verdict considers its model unlikely to be correct. In such cases, Verdict drops its model-based answer/error, and simply returns the raw answer to the user unchanged. This process is akin to hypothesis testing in statistics literatures [54].

Although no improvements are made in such cases, we take this conservative approach to ensure the correctness of our error guarantees. (See sections 3.7 and 3.11.8 for formal guarantees and empirical results, respectively).

Formally, let $t \geq 0$ be the value for which the AQP engine's answer would fall within $(\ddot{\theta}_{n+1} - t, \ddot{\theta}_{n+1} + t)$ with probability δ_v (0.99 by default) if $\ddot{\theta}_{n+1}$ were the exact answer. We call the $(\ddot{\theta}_{n+1} - t, \ddot{\theta}_{n+1} + t)$ range the likely region. To compute t, we must find the value closest to $\ddot{\theta}_{n+1}$ that satisfies the following expression:

$$\Pr\left(|X - \ddot{\theta}_{n+1}| < t\right) \geq \delta_v \tag{3.14}$$

where X is a random variable representing the AQP engine's possible answer to the new snippet if the exact answer to the new snippet was $\ddot{\theta}_{n+1}$. The AQP engine's answer can be treated as a random variable since it may differ depending on the random samples used. The probability in Equation (3.14) can be easily computed using either the central limit theorem or the Chebyshev's inequality [116]. Once the value of t is computed, Verdict rejects its model if $\tilde{\theta}_{n+1}$ falls outside the likely region $(\ddot{\theta}_{n+1} - t,\ \ddot{\theta}_{n+1} + t)$.

In summary, the pair of Verdict's improved answer and improved error, $(\hat{\theta}_{n+1}, \hat{\beta}_{n+1})$, is set to $(\ddot{\theta}_{n+1}, \ddot{\beta}_{n+1})$ if $\tilde{\theta}_{n+1}$ is within the range $(\ddot{\theta}_{n+1} - t, \ddot{\theta}_{n+1} + t)$, and is set to $(\theta_{n+1}, \beta_{n+1})$ otherwise. In either case, the error bound at confidence δ remains the same as $\alpha_\delta \cdot \hat{\beta}_{n+1}$, where α_δ is the confidence interval multiplier for probability δ.

3.10 Generalization of Verdict under Data Additions

Thus far, we have discussed our approach based on the assumption that the database is static, i.e., no tuples are deleted, added, or updated. In this section, we suggest the possibility of using Verdict even for the database that allows an important kind of data update: data append.

A naïve strategy to supporting tuples insertions would be to re-execute all past queries every time new tuples are added to the database to obtain their updated answers. This solution is obviously impractical.

Instead, Verdict still makes use of answers to past queries even when new tuples have been added since computing their answers. The basic idea is to simply lower our confidence in the raw answers of those past queries.

Assume that q_i (whose aggregate function is on A_k) is computed on an old relation r, and a set of new tuples r^a has since been added to r to form an updated relation r^u. Let $\bar{\boldsymbol{\theta}}_i^a$ be a random variable representing our knowledge of q_i's true answer on r^a, and $\bar{\theta}_i^u$ be q_i's true answer on r^u.

We represent the possible difference between A_k's values in r and those in r^a by a random variable $\boldsymbol{s}_k$ with mean μ_k and variance η_k^2. Thus:

$$\bar{\boldsymbol{\theta}}_i^a = \bar{\theta}_i + \boldsymbol{s}_k$$

The values of μ_k and η_k^2 can be estimated using small samples of r and r^a. Verdict uses the following lemma to update the raw answer and raw error for q_i:

Lemma 3.

$$E[\bar{\boldsymbol{\theta}}_i^u - \boldsymbol{\theta}_i] = \mu_k \cdot \frac{|r^a|}{|r| + |r^a|}$$

$$E[(\bar{\boldsymbol{\theta}}_i^u - \boldsymbol{\theta}_i - \frac{|r^a|\ \mu_k}{|r| + |r^a|})^2] = \beta_i^2 + \left(\frac{|r^a|\ \eta_k}{|r| + |r^a|}\right)^2$$

where $|r|$ and $|r^a|$ are the number of tuples in r and r^a, respectively.

Proof. Since we represented a snippet answer on the appended relation using a random variable $\bar{\boldsymbol{\theta}}_i^u$, we can also represent a snippet answer on the updated relation r^u using a random variable. Let the snipept answer on r^u be $\bar{\boldsymbol{\theta}}_i^u$. Then,

$$E[\bar{\boldsymbol{\theta}}_i^u - \boldsymbol{\theta}_i] = E\left[\frac{|r|\ \bar{\theta}_i}{|r| + |r^a|} + \frac{|r_a|\ \boldsymbol{s}_k}{|r| + |r^a|}\right] - \bar{\theta}_i = \frac{|r^a|\ \mu_k}{|r| + |r^a|}$$

Also,

$$\begin{aligned}
&E\left[(\bar{\boldsymbol{\theta}}_i^u - \boldsymbol{\theta}_i - \frac{|r^a|\ \mu_k}{|r| + |r^a|})^2\right] \\
&= E\left[\left(\bar{\theta}_i + \frac{|r^a|\ \boldsymbol{s}_k}{|r| + |r^a|} - \boldsymbol{\theta}_i - \frac{|r^a|\ \mu_k}{|r| + |r^a|}\right)\right] \\
&= E\left[(\bar{\theta}_i - \boldsymbol{\theta}_i)^2 + \left(\frac{|r^a|}{|r| + |r^a|}\right)^2 (\boldsymbol{s}_k - \mu_k)^2\right. \\
&\qquad\qquad \left. + 2\left(\frac{|r^a|}{|r| + |r^a|}\right)(\bar{\theta}_i - \boldsymbol{\theta}_i)(\boldsymbol{s}_k - \mu_k)\right] \\
&= \beta_i^2 + \left(\frac{|r^a|\ \eta_k}{|r| + |r^a|}\right)^2
\end{aligned}$$

where we used to independence between $(\bar{\theta}_i - \boldsymbol{\theta}_i)$ and $(\boldsymbol{s}_k - \mu_k)$. □

Once the raw answers and the raw errors of past query snippets are updated using this lemma, the remaining inference process remains the same.

3.11 Experiments

We conducted experiments to (1) quantify the percentage of real-world queries that benefit from Verdict (section 3.11.2), (2) study Verdict's average speedup and error reductions over an AQP engine (section 3.11.3), (3) test the reliability of Verdict's error bounds (section 3.11.4), (4) measure Verdict's computational overhead and memory footprint (section 3.11.5), and (5) study the impact of different workloads and data distributions on Verdict's effectiveness (section 3.11.6). In summary, our results indicated the following:

- Verdict supported a large fraction (73.7%) of aggregate queries in a real-world workload, and produced significant speedups (up to 23.0×) compared to a sample-based AQP solution.

- Given the same processing time, Verdict reduces the baseline's approximation error on average by 75.8%–90.2%.
- Verdict's runtime overhead was <10 milliseconds on average (0.02%–0.48% of total time) and its memory footprint was negligible.
- Verdict's approach was robust against various workloads and data distributions.

Moreover, section 3.11.10 shows the benefits of model-based inference in comparison to a strawman approach, which simply caches all past query answers. section 3.11.11 demonstrates Verdict's benefit for time-bound AQP engines.

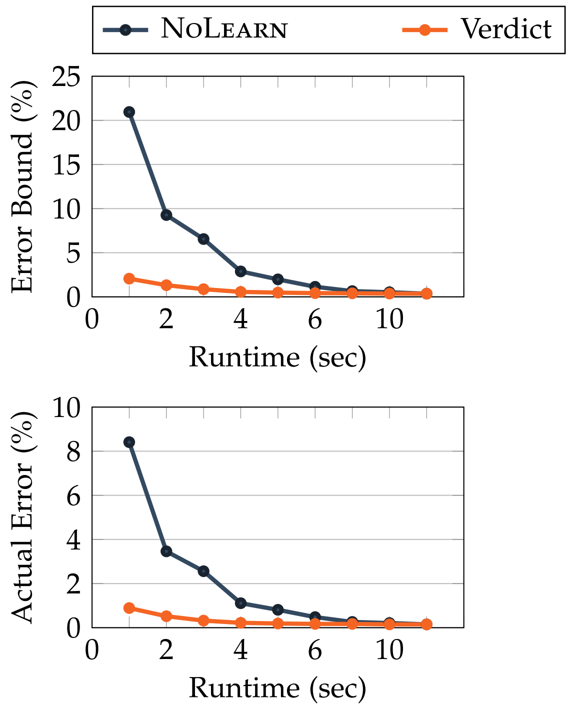
3.11.1 Experimental Setup

Datasets and Query Workloads— For our experiments, we used the three datasets described below:

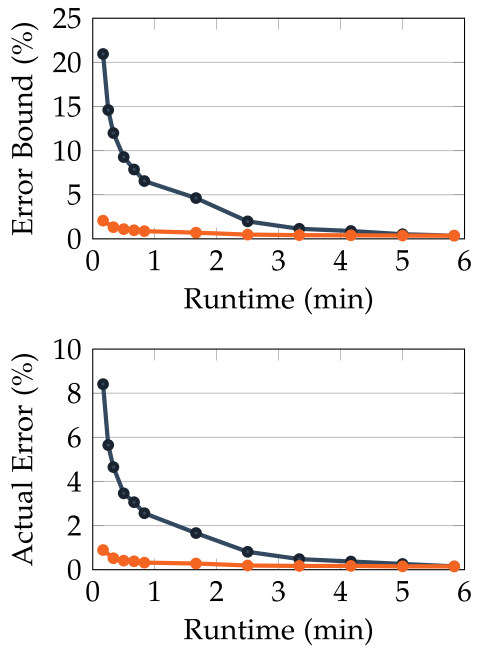
1. `Customer1`: This is a anonymized real-world query trace from one of the largest customers of a leading vendor of analytic DBMS. This dataset contains 310 tables and 15.5K timestamped queries issued between March 2011 and April 2012, 3.3K of which are analytical queries supported by Spark SQL. We did not have the customer's original dataset but had access to their data distribution, which we used to generate a 536 GB dataset.
2. `TPC-H`: This is a well-known analytical benchmark with 22 query types, 21 of which contain at least one aggregate function (including 2 queries with `min` or `max`). We used a scale factor of 100, i.e., the total data size was 100 GB. We generated a total of 500 queries using `TPC-H`'s workload generator with its default settings. The queries in this dataset include joins of up to 6 tables.
3. `Synthetic`: For more controlled experiments, we also generated large-scale synthetic datasets with different distributions (see section 3.11.6 for details).

Implementation— For comparative analysis, we implemented two systems on top of Spark SQL [21] (version 1.5.1):

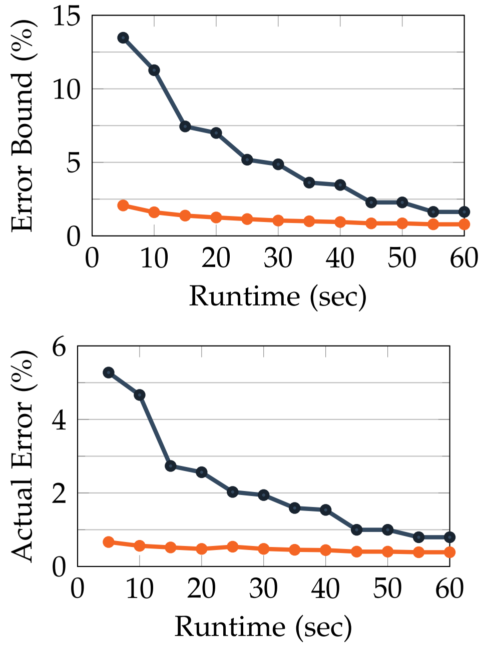
1. NoLearn: This system is an online aggregation engine that creates random samples of the original tables offline and splits them into multiple batches of tuples. To compute increasingly accurate answers to a new query, NoLearn first computes an approximate answer and its associated error bound on the first batch of tuples, and then continues to refine its answer and error bound as it processes additional batches. NoLearn estimates its errors and computes confidence intervals using closed-forms (based on the central limit theorem). Error estimation based on the central limit



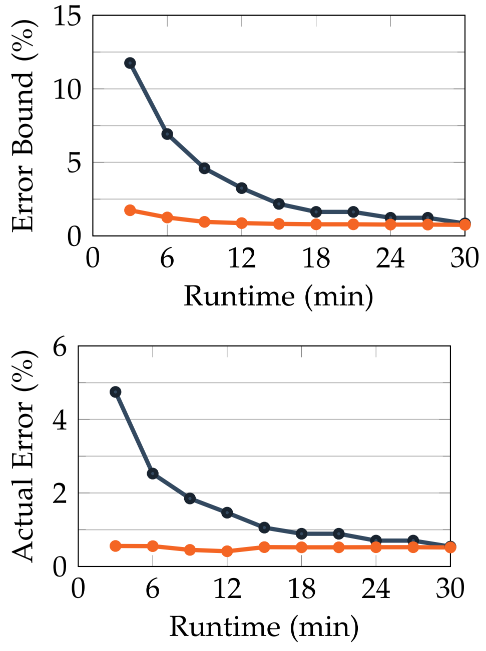
(a) Cached, `Customer1`



(b) Not Cached, `Customer1`



(c) Cached, `TPC-H`



(d) Not Cached, `TPC-H`

Figure 3.5: The relationship (i) between runtime and error bounds (top row), and (ii) between runtime and actual errors (bottom row), for both systems: NoLearn and Verdict.

Dataset	Total # of Queries with Aggregates	# of Supported Queries	Percentage
Customer1	3,342	2,463	73.7%
TPC-H	21	14	63.6%

Table 3.3: Generality of Verdict. Verdict supports a large fraction of real-world and benchmark queries.

theorem has been one of the most popular approaches in online aggregation systems [68,89,184,185] and other AQP engines [11,15,38].

2. Verdict: This system is an implementation of our proposed approach, which uses NoLearn as its AQP engine. In other words, each time NoLearn yields a raw answer and error, Verdict computes an improved answer and error using our proposed approach. Naturally, Verdict incurs a (negligible) runtime overhead, due to supported query check, query decomposition, and computation of improved answers; however, Verdict yields answers that are much more accurate in general.

Experimental Environment— We used a Spark cluster (for both NoLearn and Verdict) using 5 Amazon EC2 `m4.2xlarge` instances, each with 2.4 GHz Intel Xeon E5 processors (8 cores) and 32GB of memory. Our cluster also included SSD-backed HDFS [159] for Spark's data loading. For experiments with cached datasets, we distributed Spark's RDDs evenly across the nodes using Spark SQL DataFrame `repartition` function.

3.11.2 Generality of Verdict

To quantify the generality of our approach, we measured the coverage of our supported queries in practice. We analyzed the real-world SQL queries in `Customer1`. Among the original 15.5K queries, we used 3.3K queries that (1) were on large tables and (2) included Spark SQL-supported aggregate functions. Among those 3.3K queries, Verdict could support 2.4K queries; that is, 73.7% of the Spark SQL-supported queries could benefit from Verdict. Also, we analyzed the 21 `TPC-H` queries (that include aggregations) and found 14 queries supported by Verdict. Most of the queries that could not be support by Verdict (for both `Customer1` and `TPC-H`) were due to the min/max functions in the `select` clause, or the textual filters and disjunctions in the `where` clause. These statistics are summarized in Table 3.3. This analysis proves that Verdict can support a large class of analytical queries in practice. Next, we quantified the extent to which these supported queries benefitted from Verdict.

3.11.3 Speedup and Error Reduction

In this section, we first study the relationship between the processing time and the size of error bounds for both systems, i.e., NoLearn and Verdict. Based on this study, we then analyze Verdict's speedup and error reductions over NoLearn.

In this experiment, we used each of `Customer1` and `TPC-H` datasets in two different settings. In one setting, all samples were cached in the memories of the cluster, while in the second, the data had to be read from SSD-backed HDFS.

We allowed both systems to process half of the queries (since `Customer1` queries were timestamped, we used the first half). While processing those queries, NoLearn simply returned the query answers, but Verdict also kept the queries and their answers in its query synopsis. After processing those queries, Verdict (i) precomputed the matrix inversions and (ii) learned the correlation parameters. The matrix inversions took 1.6 seconds in total; the correlation parameter learning took 23.7 seconds for `TPC-H` and 8.04 seconds for `Customer1`. The learning process was relatively faster for `Customer1` since most of the queries included `COUNT(*)` for which each attribute did not require a separate learning. This offline training time for both workloads was comparable to the time needed for running only a single approximate query (Table 3.4).

For the second half of the queries, we recorded both systems' query processing times (i.e., runtime), approximate query answers, and error bounds. Since both NoLearn and Verdict are online aggregation systems, and Verdict produces improved answers for every answer returned from NoLearn, both systems naturally produced more accurate answers (i.e., answers with smaller error bounds) as query processing continued. Approximate query engines, including both NoLearn and Verdict, are only capable of producing expected errors in terms of error bounds. However, for analysis, we also computed the actual errors by comparing those approximate answers against the exact answers. In the following, we report their relative errors.

Figure 3.5 shows the relationship between runtime and average error bound (top row) and the relationship between runtime and average actual error (bottom row). Here, we also considered two cases: when the entire data is cached in memory and when it resides on SSD. In all experiments, the runtime-error graphs exhibited a consistent pattern: (i) Verdict produced smaller errors even when runtime was very large, and (ii) Verdict showed faster runtime for the same target errors. Due to the asymptotic nature of errors, achieving extremely accurate answers (e.g., less than 0.5%) required relatively long processing time even for Verdict.

	Cached?	Error Bound	Time Taken NoLearn	Time Taken Verdict	Speedup
Customer1	Yes	2.5%	4.34 sec	0.57 sec	**7.7**×
		1.0%	6.02 sec	2.45 sec	**2.5**×
	No	2.5%	140 sec	6.1 sec	**23.0**×
		1.0%	211 sec	37 sec	**5.7**×
TPC-H	Yes	4.0%	26.7 sec	2.9 sec	**9.3**×
		2.0%	34.2 sec	12.9 sec	**2.7**×
	No	4.0%	456 sec	72 sec	**6.3**×
		2.0%	524 sec	265 sec	**2.1**×

	Cached?	Runtime	Achieved Error Bound NoLearn	Achieved Error Bound Verdict	Error Reduction
Customer1	Yes	1.0 sec	21.0%	2.06%	**90.2%**
		5.0 sec	1.98%	0.48%	**75.8%**
	No	10 sec	21.0%	2.06%	**90.2%**
		60 sec	6.55%	0.87%	**86.7%**
TPC-H	Yes	5.0 sec	13.5%	2.13%	**84.2%**
		30 sec	4.87%	1.04%	**78.6%**
	No	3.0 min	11.8%	1.74%	**85.2%**
		10 min	4.49%	0.92%	**79.6%**

Table 3.4: Speedup and error reductions by Verdict compared to NoLearn.

Using these results, we also analyzed Verdict's speedups and error reduction over NoLearn. For speedup, we compared how long each system took until it reached a target error bound. For error reduction, we compared the lowest error bounds that each system produced within a fixed allotted time. Table 3.4 reports the results for each combination of dataset and location (in memory or on SSD).

For the `Customer1` dataset, Verdict achieved a larger speedup when the data was stored on SSD (up to 23.0×) compared to when it was fully cached in memory (7.7×). The reason was that, for cached data, the I/O time was no longer the dominant factor and Spark SQL's default overhead (e.g., parsing the query and reading the catalog) accounted for a considerable portion of the total data processing time. For `TPC-H`, on the contrary, the speedups were smaller when the data was stored on SSD. This difference stems from the different query forms between `Customer1` and `TPC-H`. The `TPC-H` dataset includes queries that join several tables, some of which are large tables that were not sampled by NoLearn. (Similar to most sample-based AQP engines, NoLearn only samples fact

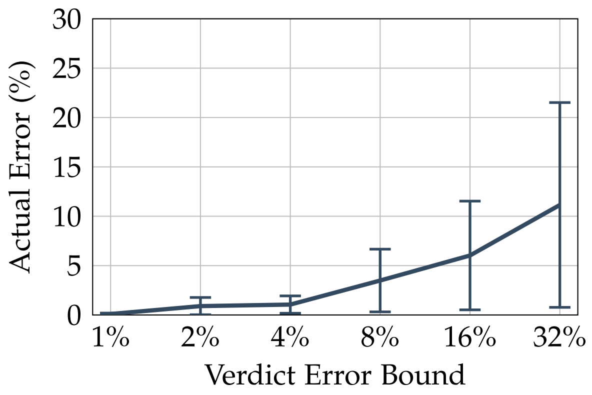


Figure 3.6: The comparison between Verdict's error bound at 95% confidence and the actual error distribution (5th, 50th, and 95th percentiles are reported for actual error distributions).

tables, not dimension tables.) Consequently, those large tables had to be read each time NoLearn processed such a query. When the data resided on SSD, loading those tables became a major bottleneck that could not be reduced by Verdict (since they were not sampled). However, on average, Verdict still achieved an impressive 6.3× speedup over NoLearn. In general, Verdict's speedups over NoLearn reduced as the target error bounds became smaller; however, even for 1% target error bounds, Verdict achieved an average of up to 5.7× speedup over NoLearn.

Table 3.4 also reports Verdict's error reductions over NoLearn. For all target runtime budgets we examined, Verdict achieved massive error reductions compared to NoLearn.

The performance benefits of Verdict depends on several important factors, such as the accuracy of past query answers and workload characteristics. These factors are further studied in sections 3.11.6 and 3.11.10.

3.11.4 Confidence Interval Guarantees

To confirm the validity of Verdict's error bounds, we configured Verdict to produce error bounds at 95% confidence and compared them to the actual errors. We ran Verdict for an amount of time long enough to sufficiently collect error bounds of various sizes.

By definition, the error bounds at 95% confidence are probabilistically correct if the actual errors are smaller than the error bounds in at least 95% of the cases. Figure 3.6 shows the 5th percentile, median, and 95th percentile of the actual errors for different sizes of error bounds (from 1% to 32%). In all cases, the 95th percentile of the actual errors were lower than the error bounds produced by Verdict, which confirms the probabilistic correctness of Verdict's error bound guarantees.

Latency	Cached	Not Cached
NoLearn	2.083 sec	52.50 sec
Verdict	2.093 sec	52.51 sec
Overhead	0.010 sec (0.48%)	0.010 sec (0.02%)

Table 3.5: The runtime overhead of Verdict.

3.11.5 Memory and Computational Overhead

In this section, we study Verdict's additional memory footprint (due to query synopsis) and its runtime overhead (due to inference). The total memory footprint of the query synopsis was 5.79MB for `TPC-H` and 18.5MB for `Customer1` workload (23.2KB per-query for `TPC-H` and 15.8KB per-query for `Customer1`). This included past queries in parsed forms, model parameters, covariance matrices, and the inverses of those covariance matrices. The size of query synopsis was small because Verdict does not retain any of the input tuples.

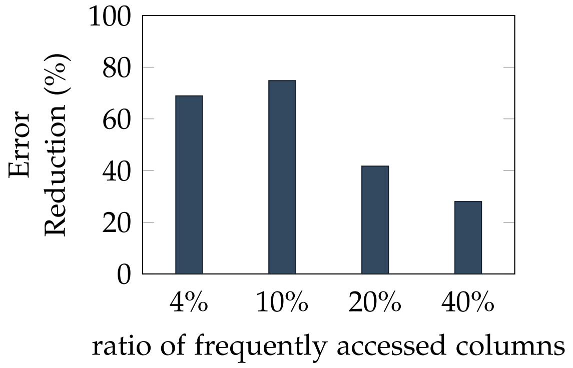
To measure Verdict's runtime overhead, we recorded the time spent for its regular query processing (i.e., NoLearn) and the additional time spent for the inference and updating the final answer. As summarized in Table 3.5, the runtime overhead of Verdict was negligible compared to the overall query processing time. This is because multiplying a vector by a $C_g \times C_g$ matrix does not take much time compared to regular query planning, processing, and network commutations among the distributed nodes. (Note that C_g=2,000 by default; see section 3.2.3.)

3.11.6 Impact of Data Distributions and Workload Characteristics

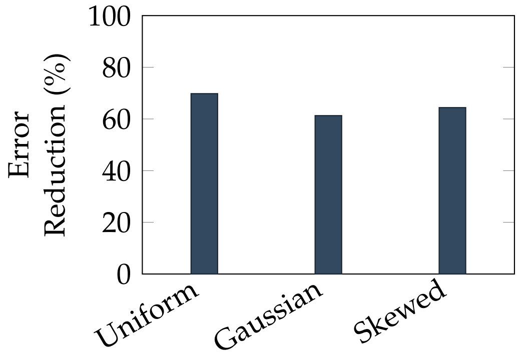
In this section, we generated various synthetic datasets and queries to fully understand how Verdict's effectiveness changes for different data distributions, query patterns, and number of past queries.

First, we studied the impact of having queries with a more diverse set of columns in their selection predicates. We produced a table of 50 columns and 5M rows, where 10% of the columns were categorical. The domains of the numeric columns were the real values between 0 and 10, and the domains of the categorical columns were the integers between 0 and 100.

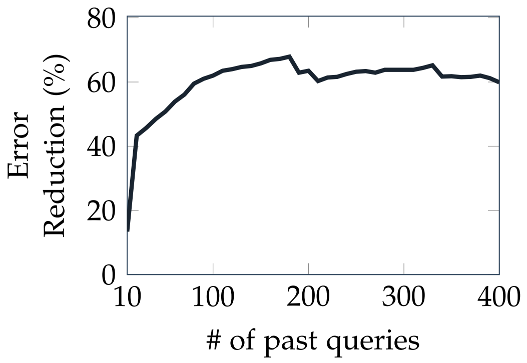
Also, we generated four different query workloads with varying proportions of frequently accessed columns. The columns used for the selection predicates were chosen according to a power-law distribution. Specifically, a fixed number of columns (called *frequently accessed columns*) had the same probability of being accessed, but the access



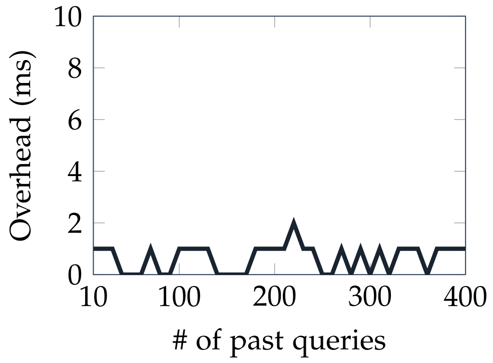
(a) Workload Diversity



(b) Data distribution



(c) Learning Behavior



(d) Overhead

Figure 3.7: The effectiveness of Verdict in reducing NoLearn's error for different (a) levels of diversity in the queried columns, (b) data distributions, and (c) number of past queries observed. Figure (d) shows Verdict's overhead for different number of past queries.

probability of the remaining columns decayed according to the power-law distribution. For instance, if the proportion of frequently accessed columns was 20%, the first 20% of the columns (i.e., 10 columns) appeared with equal probability in each query, but the probability of appearance reduced by half for every remaining column. Figure 3.7(a) shows that as the proportion of frequently accessed columns increased, Verdict's relative error reduction over NoLearn gradually decreased (the number of past queries were fixed to 100). This is expected as Verdict constructs its model based on the columns appearing in the past. In other words, to cope with the increased diversity, more past queries are needed to understand the complex underlying distribution that generated the data. Note that, according to the analytic queries in the `Customer1` dataset, most of the queries included less than 5 distinct selection predicates. However, by processing more queries, Verdict continued to learn more about the underlying distribution, and produced larger error reductions even when the workload was extremely diverse (Figure 3.7(c)).

Second, to study Verdict's potential sensitivity, we generated three tables using three different probability distributions: uniform, Gaussian, and a log-normal (skewed) distribution. Figure 3.7(b) shows Verdict's error reductions when queries were run against each table. Because of the power and generality of the maximum entropy principle taken by Verdict, it delivered a consistent performance irrespective of the underlying distribution.

Third, we varied the number of past queries observed by Verdict before running our test queries. For this study, we used a highly diverse query set (its proportion of frequently accessed columns was 20%). Figure 3.7(c) demonstrates that the error reduction continued increasing as more queries were processed, but its increment slowed down. This is because, after observing enough information, Verdict already had a good knowledge of the underlying distribution, and processing more queries barely improved its knowledge. This result indicates that Verdict is able to deliver reasonable performance without having to observe too many queries.

This is because, after observing enough information, Verdict already has a good knowledge of the underlying distribution, and processing more queries barely improves its knowledge. This result indicates that Verdict is able to deliver reasonable performance without having to observe too many queries.

Lastly, we studied the negative impact of increasing the number of past queries on Verdict's overhead. Since Verdict's inference consists of a small matrix multiplication, we did not observe a noticeable increase in its runtime overhead even when the number of queries in the query synopsis increased (Figure 3.7(d)).

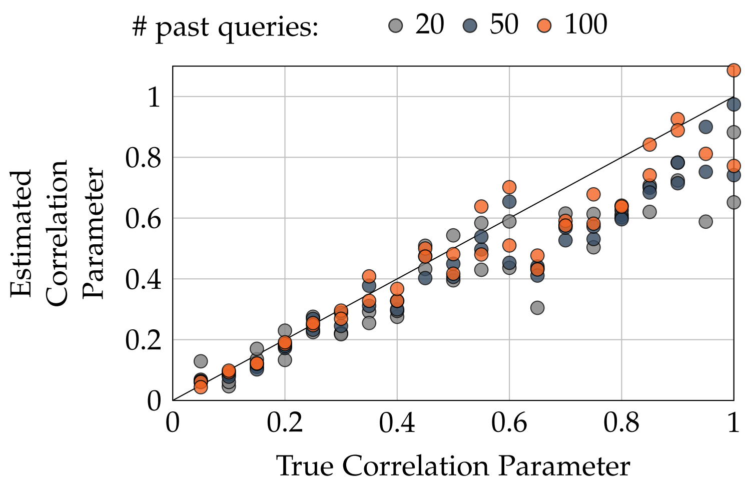


Figure 3.8: Correlation Parameter Learning

Recall that the domain size of the attributes does not affect Verdict's computational cost since only the lower and upper bounds of range constraints are needed for covariance computations (section 3.4.2).

3.11.7 Accuracy of Parameter Learning

In this section, we demonstrate our empirical study on the effectiveness of Verdict's correlation parameter estimation process. For this, we used the synthetic datasets generated from pre-determined correlation parameters to see how close Verdict could estimate the values of those correlation parameters. We let Verdict estimate the correlation parameter values using three different numbers of past snippets (20, 50, and 100) for various datasets with different correlation parameter values.

Figure 3.8 shows the results. In general, the correlation parameter values discovered by Verdict's estimation process were consistent with the true correlation parameter values. Also, the estimated values tended to be closer to the true values when a larger number of past snippets were used for the estimation process. This result indicates that Verdict can effectively learn statistical characteristics of the underlying distribution just based on the answers to the past queries.

3.11.8 Model Validation

This section studies the effect of Verdict's model validation described in section 3.9. For this study, we first generated synthetic datasets with several predetermined correlation parameters values. Note that one can generate such synthetic datasets by first determin-

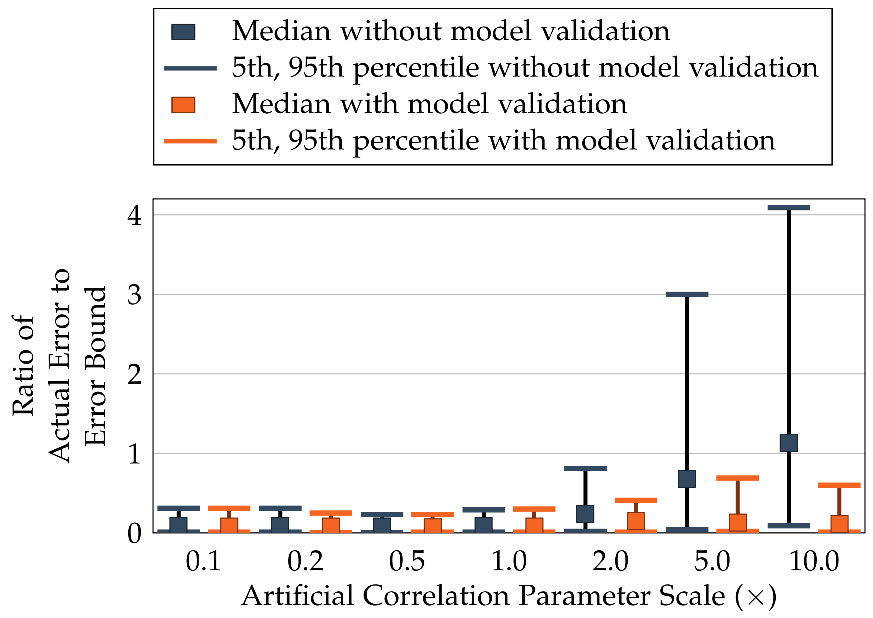


Figure 3.9: Effect of model validation. For Verdict's error bounds to be correct, the 95th percentile should be below 1.0. One can find that, with Verdict's model validation, the improved answers and the improved errors were probabilistically correct even when largely incorrect correlation parameters were used.

ing a joint probabilistic distribution function with predetermined correlation parameter values and sampling attribute values from the joint probability distribution function. In usual usage scenario, Verdict estimates those correlation parameters from past snippets; however, in this section, we manually set the values for the correlation parameters in Verdict's model, to test the behavior of Verdict running with possibly incorrect correlation parameter values.

Figure 3.9 reports the experiment results from when Verdict was tested without a model validation step and with a model validation step, respectively. In the figure, the values on the X-axis are artificial correlation parameter scales, i.e., the product of the true correlation parameters and each of those scales are set in Verdict's model. For instance, if a true correlation parameter was 5.0, and the "artificial correlation parameter scale" was 0.2, Verdict's model was set to 1.0 for the correlation parameter. Thus, the values of the correlation parameters in Verdict's model were set to the true correlation parameters, when the "artificial correlation parameter scale" was 1.0. Since the Y-axis reports the ratio of the actual error to Verdict's error bound, Verdict's error bound was correct when the value on the Y-axis was below 1.0.

In the figure, one can observe that, Verdict, used without the model validation, produced incorrect error bounds when the correlation parameters used for the model devi-

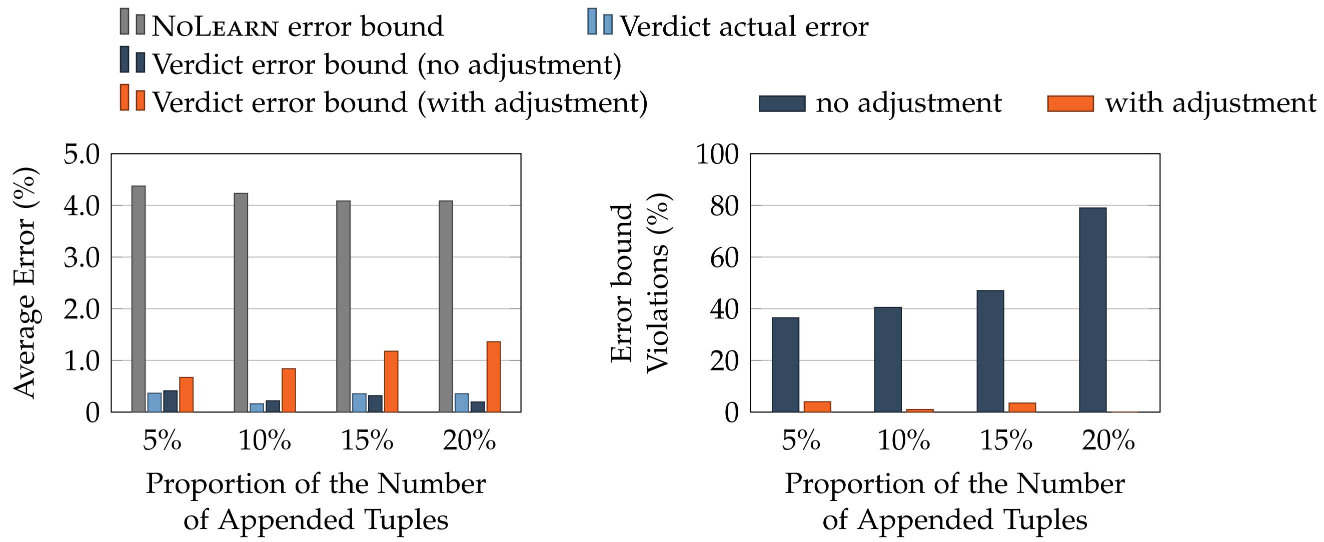


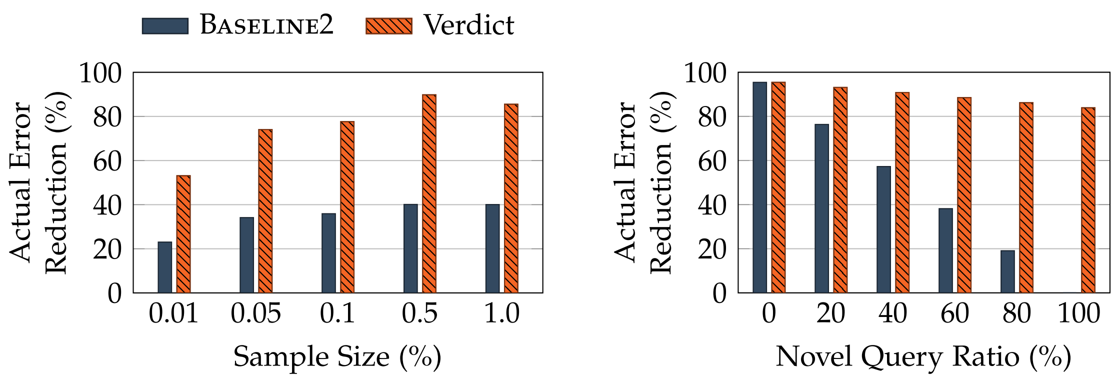
Figure 3.10: Data append technique (section 3.10) is highly effective in delivering correct error estimates in face of new data.

ated largely from the true correlation parameter values. However, Verdict's model validation could successfully identify incorrect model-based answers and provide correct error bounds by replacing those incorrect model-based answers with the raw answers computed by the AQP system.

3.11.9 Data Append

In this section, we empirically study the impact of new data (i.e., tuple insertions) on Verdict's effectiveness. We generated an initial synthetic table with 5 million tuples and appended additional tuples to generate different versions of the table. The newly inserted tuples were generated such that their attribute values gradually diverged from the attribute values of the original table. We distinguish between these different versions by the ratio of their newly inserted tuples, e.g., a 5% appended table means that 250K (5% of 5 million) tuples were added. We then ran the queries and recorded the error bounds of VerdictAdjust and VerdictNoAdjust (our approach with and without the technique introduced in section 3.10). We also measured the error bounds of NoLearn and the actual error.

As shown in Figure 3.10(a), VerdictNoAdjust produced overly-optimistic error bounds (i.e., lower than the actual error) for 15% and 20% appends, whereas VerdictAdjust produced valid error bounds in all cases. Since this figure shows the *average* error bounds across all queries, we also computed the fraction of the individual queries for which each method's error bounds were violated. In Figure 3.10(b), the Y-axis indicates those cases



(a) Sample Sizes for Past Queries **(b)** Query Workload Composition

Figure 3.11: (a) Comparison of Verdict and Baseline2 for different sample sizes used by past queries and (b) comparison of Verdict and Baseline2 for different ratios of novel queries in the workload.

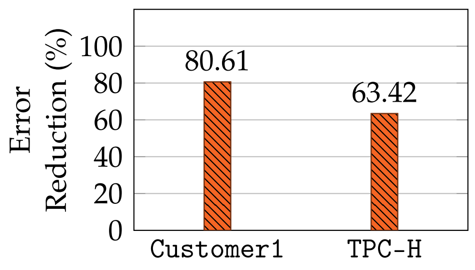
where the actual error was larger than the system-produced error bounds. This figure shows more error violations for VerdictNoAdjust, which increased with the number of new tuples. In contrast, VerdictAdjust produced valid error bounds in most cases, while delivering substantial error reductions compared to NoLearn.

3.11.10 Verdict vs. Simple Answer Caching

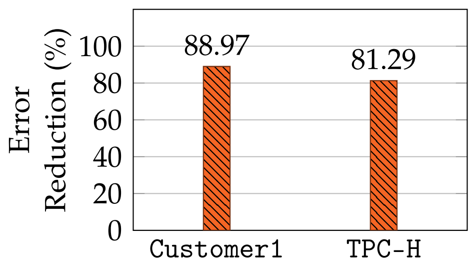
To study the benefits of Verdict's model-based inference, we consider another system Baseline2, and make comparisons between Verdict and Baseline2, using the `TPC-H` dataset. Baseline2 is similar to NoLearn but returns a cached answer if the new query is identical to one of the past ones. When there are multiple instances of the same query, Baseline2 caches the one with the lowest expected error.

Figure 3.11(a) reports the average actual error reductions of Verdict and Baseline2 (over NoLearn), when different sample sizes were used for past queries. Here, the same samples were used for new queries. The result shows that both systems' error reductions were large when large sample sizes were used for the past queries. However, Verdict consistently achieved higher error reductions compared to Baseline2, due to its ability to benefit novel queries as well as repeated queries (i.e., the queries that have appeared in the past).

Figure 3.11(b) compares Verdict and Baseline2 by changing the ratio of novel queries in the workload. Understandably, both Verdict and Baseline2 were more effective for workloads with fewer novel queries (i.e., more repeated queries); however, Verdict was also effective for workloads with many novel queries.



(a) Cached



(b) Not Cached

Figure 3.12: Average error reduction by Verdict (compared to NoLearnTime) for the same time budget.

3.11.11 Error Reductions for Time-Bound AQP Engines

In this section, we show Verdict's error reductions over a time-bound AQP system. First, we describe our experiment setting. Next, we present our experiment results.

Setup— Here, we describe two systems, NoLearnTime and Verdict, which we compare in this section:

1. NoLearnTime: This system runs queries on *samples* of the original tables to obtain fast but approximate query answers and their associated estimated errors. This is the same approach taken by existing AQP engines, such as [10, 15, 38, 144, 148, 186]. Specifically, NoLearnTime maintains uniform random samples created offline (10% of the original tables), and it uses the largest samples that are small enough to satisfy the requested time bounds.
2. Verdict: This system invokes NoLearnTime to obtain raw answers/errors but modifies them to produce improved answers/errors using our proposed inference process. Verdict translates the user's requested time bound into an appropriate time bound for NoLearnTime.

Experiment Results— This section presents the error reduction by Verdict compared to NoLearnTime. For experiments, we ran the same set of queries as in section 5.6 with both Verdict and NoLearnTime described above. For comparison, we used the identical time-bounds for both Verdict and NoLearnTime. Specifically, we set the time-bounds as 2 seconds and 0.5 seconds for the `Customer1` and `TPC-H` datasets cached in memory, respectively; and we set the time-bounds as 5.0 seconds for both `Customer1` and `TPC-H` datasets loaded from SSD. Figure 3.12 reports Verdict's error reductions over NoLearnTime for each of four different combinations of a dataset and cache setting.

In Figure 3.12, one can observe that Verdict achieved large error reductions (63%–86%) over NoLearnTime. These results indicate that the users of Verdict can obtain

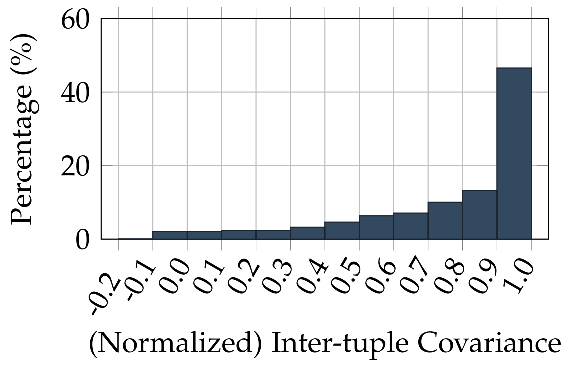


Figure 3.13: *Inter-tuple Covariances* for 16 real-life UCI datasets.

much more precise answers compared to the users of NoLearnTime within the same time-bounds.

3.12 Prevalence of Inter-tuple Covariances in Real-World

In this section, we demonstrate the existence of the inter-tuple covariances in many real-world datasets by analyzing well-known datasets from the UCI repository [107]. We analyzed the following well-known 16 datasets: cancer, glass, haberman, ionosphere, iris, mammographic-masses, optdigits, parkinsons, pima-indians-diabetes, segmentation, spambase, steel-plates-faults, transfusion, vehicle, vertebral-column, and yeast.

We extracted numeric attributes (or equivalently, columns) from those datasets and composed each of the datasets into a relational table. Suppose a dataset has m attributes. Then, we computed the correlation between adjacent attribute values in the i-th column when the column is sorted in order of another j-th column—i and j are the values (inclusively) between 1 and m, and $i \neq j$. Note that there are $m(m-1)/2$ number of pairs of attributes for a dataset with m attributes. We analyzed all of those pairs for each of 16 datasets listed above.

Figure 3.13 shows the results of our analysis. The figure reports the percentage of different levels of correlations (or equivalently, normalized inter-tuple covariances) between adjacent attributes. One can observe that there existed strong correlations in the datasets we analyzed. Remember that the users of Verdict do not need to provide any information regarding the inter-tuple covariances; Verdict automatically detects them as described in section 3.8, relying on the past snippet answers stored in the query synopsis.

3.13 Technical Details

In this section, we present the mathematical details we have omitted in the main body of this chapter. First, we describe the analytics expression for the double-integrals in Equation (3.10). Second, we extend the result of section 3.4 to categorical attributes. Third, we provides details on some correlation parameter computations.

3.13.1 Double-integration of Exp Function

For the analytic computation of Equation (3.10), we must be able to analytically express the result of the following double integral:

$$f(x,y) = \int_a^b \int_c^d \exp\left(-\frac{(x-y)^2}{z^2}\right) dx\, dy \tag{3.15}$$

To obtain its indefinite integral, we used the Symbolic Math Toolbox in Matlab, which yielded:

$$f(x,y) = -\frac{1}{2}\left(z^2 \exp\left(-\frac{(x-y)^2}{z^2}\right)\right) - \frac{\sqrt{\pi}}{2} z(x-y)\,\mathrm{erf}\left(\frac{x-y}{z}\right)$$

where $\mathrm{erf}(\cdot)$ is the *error function*, available from most mathematics libraries. Then, the definite integral in Equation (3.15) is obtained by $f(b,d) - f(b,c) - f(a,d) + f(a,c)$.

3.13.2 Handling Categorical Attributes

Thus far, we have assumed that all dimension attributes are numeric. This section describes how to handle dimension attributes that contain both numeric and categorical attributes. Let a tuple $\boldsymbol{t} = (a_1, \ldots, a_c, a_{c+1}, \ldots, a_l)$, where c is the number of categorical attributes; the number of numeric attributes is $l - c$. Also $\boldsymbol{t}_c = (a_1, \ldots, a_c)$ and $\boldsymbol{t}_l = (a_{c+1}, \ldots, a_l)$. The covariance between two query snippet answers in Equation (3.8) is extended as:

$$\sum_{\boldsymbol{t}_c \in F_i^c} \sum_{\boldsymbol{t}'_c \in F_j^c} \int_{\boldsymbol{t}_l \in F_i^l} \int_{\boldsymbol{t}'_l \in F_j^l} \mathrm{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}'))\, d\boldsymbol{t}\, d\boldsymbol{t}'$$

where F_i^c is the set of $\boldsymbol{t}_c$ that satisfies q_i's selection predicates on categorical attributes, and F_i^l is the set of $\boldsymbol{t}_l$ that satisfies q_i's selection predicates on numeric attributes. The first question, then, is how to define the inter-tuple covariance, i.e., $\mathrm{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}'))$, when two arbitrary tuples $\boldsymbol{t}$ and $\boldsymbol{t}'$ follow the schema with both categorical and numeric attributes. For this, Verdict extends the previous inter-tuple covariance in Equation (3.9), which was defined only for numeric attributes, as follows:

$$\operatorname{cov}(\nu_g(\boldsymbol{t}), \nu_g(\boldsymbol{t}')) \approx \sigma_g^2 \cdot \prod_{k=1}^{c} \delta(a_k, a_k') \prod_{k=c+1}^{l} \exp\left(-\frac{(a_k - a_k')^2}{l_{g,k}^2}\right)$$

where $\delta(a, a')$ returns 1 if $a = a'$ and 0 otherwise. The inter-tuple covariance between two tuples become zero if they include different categorical attribute values. Note that this is a natural choice, since the covariance between the two random variables, independently and identically drawn from the same distribution, is zero. With the above definition of the inter-tuple covariance, $\operatorname{cov}(\bar{\boldsymbol{\theta}}_i, \bar{\boldsymbol{\theta}}_j)$ is expressed as:

$$\begin{aligned} &\sigma_g^2 \prod_{k=1}^{c} |F_{i,k} \cap F_{j,k}| \\ &\prod_{k=c+1}^{l} \int_{s_{i,k}}^{e_{i,k}} \int_{s_{j,k}}^{e_{j,k}} \exp\left(-\frac{(a_k - a_k')^2}{l_{g,k}^2}\right) da_k' a_k \end{aligned} \tag{3.16}$$

where $F_{i,k}$ and $F_{j,k}$ are the set of the A_k's categorical attribute values used for the `in` operator in i-th and j-th query snippet, respectively. If q_i includes a single equality constraint for a categorical attribute A_k, e.g., A_k `= 1`, the equality constraint is conceptually treated as the `in` operator with the list including only that particular attribute value. If no constraints are specified in q_i for a categorical attribute A_k, $F_{i,k}$ is conceptually treated as a universal set including all attribute values in A_k. The above expression can be computed efficiently, since counting the number of common elements between two sets can be performed in a linear time using a hash set, and the double integral of an exponential function can be computed analytically (section 3.13.1).

3.13.3 Analytically Computed Parameter Values

While Verdict learns correlation parameters, i.e., $l_{g,1}, \ldots, l_{g,l}$, by solving an optimization problem, the two other parameters, i.e., the expected values of query answers (namely $\vec{\mu}$) and the multiplier for $\rho_g(\boldsymbol{t}, \boldsymbol{t}')$ (namely σ_g), are analytically computed as follows. Recall that $\vec{\mu}$ is used for computing model-based answers and errors (Equation (3.12)), and σ_g is used for computing the covariances between pairs of query answers (Equation (3.16)).

First, we use a single scalar value μ for the expected values of the prior distribution; that is, every element of $\vec{\mu}$ is set to μ once we obtain the value. Note that it only serves as the means in the prior distribution. We take different approaches for `AVG(`A_k`)` and

FREQ(*) as follows. For AVG(*), we simply set $\mu = \sum_{i=1}^{n} \theta_i / n$; whereas, for FREQ(*), we set $\mu = \sum_{i=1}^{n} \theta_i / |F_i|$ where $|F_i|$ is the area of the hyper-rectangle $\prod_{k=c+1}^{l} (s_{i,k}, e_{i,k})$ specified as q_i's selection predicates on numeric attributes.

Second, observe that σ_g^2 is equivalent to the variance of $\nu_g(\boldsymbol{t})$. For AVG(A_k), we use the variance of $\theta_1, \ldots, \theta_n$; for FREQ(*), we use the variance of $\theta_1 / |F_i|, \ldots, \theta_n / |F_i|$. We attempted to learn the optimal value for σ_g^2 in the course of solving the optimization problem (Equation (3.13)); however, the local optimum did not produce a model close to the true distribution.

3.14 Related Work

Approximate Query Processing— There has been substantial work on sampling-based approximate query processing [11,12,15,26,38,41,49,56,70,89,121,131,145,160,171]. Some of these systems differ in their sample generation strategies. Some of these systems differ in their sample generation strategies (see [127] and the references within). For instance, STRAT [38] and AQUA [12] create a single stratified sample, while BlinkDB creates samples based on *column sets*. Online Aggregation (OLA) [40, 68, 134, 177] continuously refines its answers during query execution. Others have focused on obtaining faster or more reliable error estimates [14, 178]. These are orthogonal to our work, as reliable error estimates from an underlying AQP engine will also benefit DBL. There is also AQP techniques developed for specific domain, e.g., sequential data [20, 140], probabilistic data [59, 132], and RDF data [72, 164], and searching in high-dimensional space [137]. However, our focus in this chapter is on general (SQL-based) AQP engines.

Adaptive Indexing, View Selection— Adaptive Indexing and database cracking [73,74, 141] has been proposed for a column-store database as a means of incrementally updating indices as part of query processing; then, it can speed up future queries that access previously indexed ranges. While the adaptive indexing is an effective mechanism for exact analytic query processing in column-store databases, answering queries that require accessing multiple columns (e.g., selection predicates on multiple columns) is still a challenging task: column-store databases have to join relevant columns to reconstruct tuples. Although Idreos et al. [74] pre-join some subsets of columns, the number of column combinations still grows exponentially as the total number of columns in a table increases. Verdict can easily handle queries with multiple columns due to its analytic inference. Materialized views are another means of speeding up future aggregate queries [48, 65, 85, 126]. Verdict also speed up aggregate queries, but Verdict does not require strict query containments as in materialized views.

Pre-computation— COSMOS [177] stores the results of past queries as multi-dimensional cubes, which are then reused if they are contained in the new query's input range, while boundary tuples are read from the database. This approach is not probabilistic and is limited to low-dimensional data due to the exponential explosion in the number of possible cubes. Also, similar to view selection, COSMOS relies on strict query containment.

Model-based and Statistical Databases— Statistical approaches have been used in databases for various goals. MauveDB [46] constructs views that express a statistical model, hiding the possible irregularities of the underlying data. MauveDB's goal is to support statistical modeling, such as regression or interpolation. , rather than speeding up future query processing. BayesDB [119] provides a SQL-like language that enables non-statisticians to declaratively use various statistical models. Bayesian networks have been used for succinctly capturing correlations among attributes [60]. Exploiting these correlations can be an interesting future direction for DBL.

Maximum Entropy Principle— In the database community, the principle of maximum entropy (ME) has been previously used for determining the most surprising piece of information in a data exploration context [152], and for constructing histograms based on cardinality assertions [92]. Verdict uses ME differently than these previous approaches; they assign a unique variable to each non-overlapping area to represent the number of tuples belonging to that area. This approach poses two challenges when applied to an AQP context. First, it requires a slow iterative numeric solver for its inference. Thus, using this approach for DBL may eliminate any potential speedup. Second, introducing a unique variable for every non-overlapping area can be impractical as it requires $O(2^n)$ variables for n past queries. Finally, the previous approach cannot express inter-tuple covariances in the underlying data. In contrast, Verdict's approach handles arbitrarily overlapping ranges in multidimensional space with $O(n)$ variables (and $O(n^2)$ space), and its inference can be performed analytically.

3.15 Summary

In this chapter, we presented database learning (DBL), a novel approach to exploit past queries' (approximate) answers for speeding up new queries using a principled statistical methodology. We presented a prototype system of this vision, called Verdict, implemented on top of Spark SQL. Through extensive experiments on real-world and benchmark query logs, we demonstrated that Verdict supported 73.7% of real-world analytical queries, speeding them up by up to 23$\times$ compared to an online aggregation AQP engine.

Here, we finish the first part of this dissertation. Although our contribution in this part is limited to showing that exploiting past computations can speed up the analytic SQL queries, our future work aims to exploit past computations for speeding up general SQL query processing. Chapter 6 describes more details on this future work.

Part II
Building Task-aware Synopses

Chapter 4

Accurate Approximate Searching by Learning from Data

Recall that this dissertation presents two approaches—exploiting past computations and building task-aware synopses—for improving approximate query processing (AQP). In this chapter, we present one of the techniques that speeds up AQP by building task-aware synopses. This technique speeds up an essential task for many data mining algorithms: similarity search.

4.1 Motivation

Finding the k most similar data items to a user's query item (known as k-nearest neighbors or kNN) is a common building block of many important applications. In machine learning, fast kNN techniques boost the classification speed of non-parametric classifiers [33, 95, 155, 188]. kNN is also a crucial step in collaborative filtering, a widely used algorithm in online advertisement and movie/music recommendation systems [97, 154]. Moreover, in databases and data mining, finding the k-most similar items to users' queries is the crux of image and text searching [106, 163, 167].

Unfortunately, despite thirty years of research in this area [23, 44, 64, 76, 90, 108, 180], *exact* kNN queries are still prohibitively expensive, especially over high-dimensional objects, such as images, audio, videos, documents or massive scientific arrays [173]. For example, one of the most recent approaches for exact kNN that exploits sophisticated pruning is only $9\times$ faster than a baseline table scan [90]. This prohibitive cost has given rise to *approximate* kNN.

One of the most common approaches to finding *approximate* kNN is using a set of binary hash functions that map each data item into a binary vector, called a hashcode. The database then finds the kNN items using the Hamming distance[1] among these (binary) hashcodes. Due to special properties of these binary vectors [130], searching for kNN in the hash space (a.k.a. Hamming space) can be performed much more efficiently than the original high-dimensional space. These approaches are approximate because a query item's kNN in the Hamming space may be different than its kNN in the original space. Thus, the accuracy of hashing-based approximations is judged by their effectiveness in preserving the kNN relationships (among the original items) in the Hamming space. In other words, given a query item q and its kNN set $\{v_1, \cdots, v_k\}$, a hash function $\mathbf{h}$ should be chosen such that most of the hashcodes in $\{\mathbf{h}(v_1), \cdots, \mathbf{h}(v_k)\}$ fall in the kNN set of $\mathbf{h}(q)$ in the Hamming space.

Existing Approaches— Starting with the pioneering work of Gionis, Indyk, and Motwani on *locality sensitive hashing (LSH)* over 15 years ago [61], numerous techniques have been proposed to improve the accuracy of the hashing-based kNN procedures [18, 37, 45, 63, 66, 69, 83, 84, 99, 100, 109, 113, 149, 174, 179]. Implicitly or explicitly, almost all hashing algorithms pursue the following goal: to preserve the relative distance of the original items in the Hamming space. That is, if we denote the distance between items v_1 and v_2 by $\|v_1 - v_2\|$ and the Hamming distance between their respective hashcodes by $\|\mathbf{h}(v_1) - \mathbf{h}(v_2)\|_H$, the hash function $\mathbf{h}$ is chosen such that the value of $\|\mathbf{h}(v_1) - \mathbf{h}(v_2)\|_H$ is (ideally) a linear function of $\|v_1 - v_2\|$, as shown in Figure 5.1(c). Thus, while previous techniques use different ideas, they tend to minimize the Hamming distance of nearby items while maximizing it for far apart items.

Figure 5.1(a) uses a toy example with nine data items (including a query item q) to illustrate how existing methods choose their hashcodes. In this example, four hyperplanes h_1, h_2, h_3, and h_4 are used to generate 4-bit hashcodes. Here, each hyperplane h_i acts as a binary separator to determine the i-th leftmost bit of the output hashcode; the i'th bit of $\mathbf{h}(x)$ is 0 if the item x falls on the left side of h_i and this bit is 1 otherwise. For instance, the hashcode for v_3 is 1000 because v_3 falls in the region to the right of h_1 and to the left of h_2, h_3 and h_4 hyperplanes.

To preserve the original distances, existing hashing techniques tend to place fewer (more) separators between nearby (far apart) items to ensure that their hashcodes differ in fewer (more) positions and have a smaller (larger) Hamming distance. In other words, the expected number of separators between a pair of items tends to be roughly proportional to their relative distance in the original space.[2] In this example, v_5's distance

[1]Hamming distance between two binary vectors of equal length is the number of positions at which the corresponding bits are different.

[2]We provide a more precise dichotomy of previous work in Section 4.5.

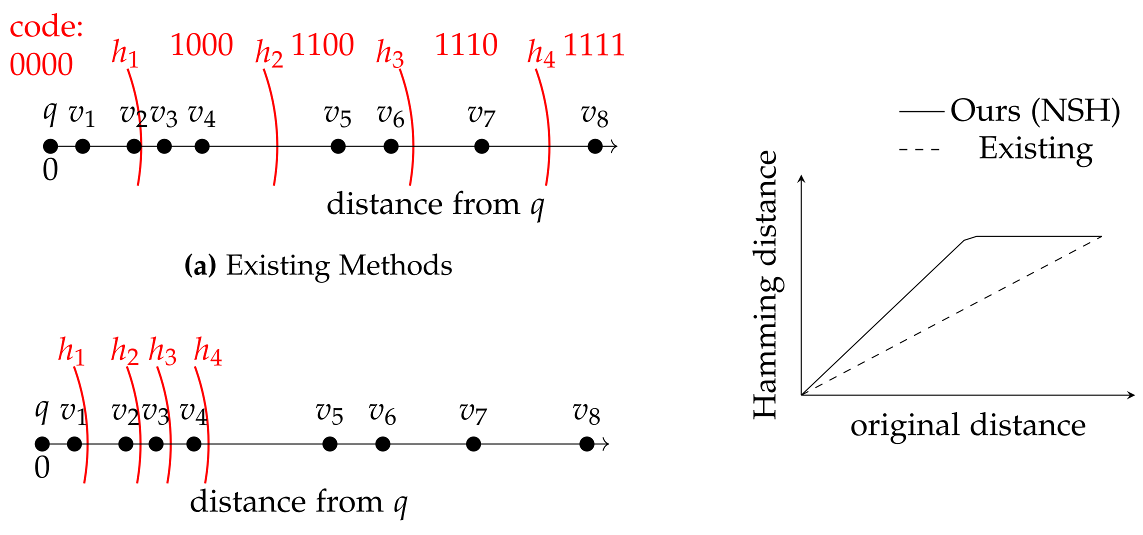
from q is twice v_4's distance from q. This ratio remains roughly the same after hashing: $\|\mathbf{h}(v_5) - \mathbf{h}(q)\|_H = \|1100 - 0000\|_H = 2$ while $\|\mathbf{h}(v_4) - \mathbf{h}(q)\|_H = \|1000 - 0000\|_H = 1$. Likewise, since v_8 is the farthest item from q, four separators are placed between them, causing their hashcodes to differ in four positions (0000 versus 1111) and thus yielding a greater Hamming distance, namely $\|\mathbf{h}(v_8) - \mathbf{h}(q)\|_H = 4$.

This goal is intuitive and can capture the intra-item similarities well. However, this approach requires a large number of hash bits (i.e., separators) to accurately capture all pair-wise similarities. Since using longer hashcodes increases the response time of search operations, we need a better strategy than simply increasing the number of separators. In this chapter, we make the following observation. Since the ultimate goal of the hashing phase is to simply find the kNN items, preserving all pair-wise similarities is unnecessary and wasteful. Rather, we propose to spend our hash bits on directly maximizing the accuracy of the kNN task itself, as described next.

Our Approach— In this work, we pursue the opposite goal of previous approaches. Instead of preserving the proximity of similar items in the Hamming space, we maximize their Hamming distance. In other words, instead of placing fewer separators between nearby items and more between far apart items, we do the opposite (compare Figures 5.1(a) and (b)).

We argue that this seemingly counter-intuitive idea is far more effective at solving the kNN problem, which is the ultimate goal of hashing. The key intuition is that we should not use our limited hash bits on capturing the distances among far apart items. Instead, we use our hash bits to better distinguish nearby items, which are likely to be in each other's kNN sets. Given a fixed number of hash bits (i.e., separators), we can achieve this distinguishing power by placing more separators among similar items. In the previous example, to find the 3-NN (i.e., $k = 3$) items for item q, we must be able to accurately compare v_3 and v_4's distances to q using their respective hashcodes. In other words, we need to have $\|\mathbf{h}(v_3) - \mathbf{h}(q)\|_H < \|\mathbf{h}(v_4) - \mathbf{h}(q)\|_H$ in order to infer that $\|v_3 - q\| < \|v_4 - q\|$.

However, due to the proximity of v_3 and v_4, existing methods are likely to assign them the same hashcode, as shown in Figure 5.1(a). In contrast, our strategy has a higher chance of correctly differentiating v_3 and v_4, due to its higher number of separators among nearby items. This is shown in Figure 5.1(b), v_3 and v_4's hashcodes differ by one bit. Obviously, our idea comes at the cost of confusing far apart items. As shown in Figure 5.1(b), we will not be able to differentiate q's distance to any of v_5, v_6, v_7, or v_8. However, this is acceptable if the user is interested in $k \leq 4$.



(a) Existing Methods

(b) Our Method

(c)

Figure 4.1: In (a) and (b), the vertical arcs indicate the boundaries where the Hamming distance from q increases by 1. The third figure (c) shows the relationship between data items' original distance and their *expected* Hamming distance.

This intuition can be applied to more general cases, where the query item q or the value of k may not be necessarily known in advance. If we choose a neighborhood size just large enough to include most of the kNN items that typical users are interested in, e.g., for $k = 1$ to 1000, we expect an increased accuracy in correctly ordering such items, and hence returning the correct kNN items to the user. Since the value of k is typically much smaller than the total number of the items in a database, we expect significant gains over existing techniques that seek to preserve all pair-wise distances using a fixed number of hash bits.

The stark difference between our strategy and previous techniques is summarized in Figure 5.1(c). The goal of existing methods is to assign hashcodes such that the Hamming distance between each pair of items is as close to a linear function of their original distance as possible. Our method changes the shape of this function, shown as a solid line; we impose a larger slope when the original distance between a pair of items is small, and allow the curve to level off beyond a certain point. This translates to a higher probability of separating the kNN items from others in our technique (we formally prove this in Section 4.3.1).

The main challenge then is how to devise a hashing mechanism that can achieve this goal. We solve this problem by proposing a special transformation that stretches out the distance between similar items (compared to distant items). Our method, called

Neighbor-Sensitive Hashing (NSH), uses these transformed representations of items to achieve the goal described above.

Contributions— In this chapter, we make several contributions.

1. We formally prove that increasing the distance between similar items in the Hamming space increases the probability of successful identification of *k*NN items (Section 4.3.1).
2. We introduce *Neighbor-Sensitive Hashing (NSH)*, a new hashing algorithm motivated by our seemingly counter-intuitive idea (Sections 4.3.2, 4.3.3, and 4.3.4).
3. We confirm our formal results through extensive experiments, showing the superiority of *Neighbor-Sensitive Hashing* over *Locality-Sensitive Hashing* [18] and other state-of-the-art hashing algorithms for approximate *k*NN [57, 69, 83, 109, 113, 174]. (Section 5.6).

In summary, our algorithm for NSH achieves 250× speedup over the baseline, obtaining an average recall of 57.5% for 10-NN retrieval tasks. Compared to the state-of-the-art hashing algorithms, our algorithm reduces the search time by up to 34% for a fixed recall (29% on average), and improves the recall by up to 31% for a fixed time budget.[3]

We overview the end-to-end workflow of hashing-based techniques in Section 4.2. We present our NSH strategy in Section 4.3. Section 5.6 reports our empirical analysis and comparisons against existing hashing algorithms. Section 4.5 overviews the related work, and Section 4.6 concludes our chapter with future work.

4.2 Hashing-based kNN Search

In this section, we provide the necessary background on hashing-based approximate *k*NN. Section 4.2.1 explains a typical workflow in hashing-based approximate *k*NN. Section 4.2.2 reviews a well-known principle in designing hash functions to compare with ours.

4.2.1 Workflow

Figure 4.2 summarizes the prototypical workflow of a hashing-based *k*NN system. Among the three componenets, Hash Function and Hamming Search are more important. Hash Function is the component that converts the data items residing in the database at index time into binary vectors, known as *hashcodes*. The same function is used to also convert

[3]In approximate *k*NN, a simple post ranking step is used to mitigate the impact of low precision while preserving the recall [79, 80]. See Section 4.4.1.

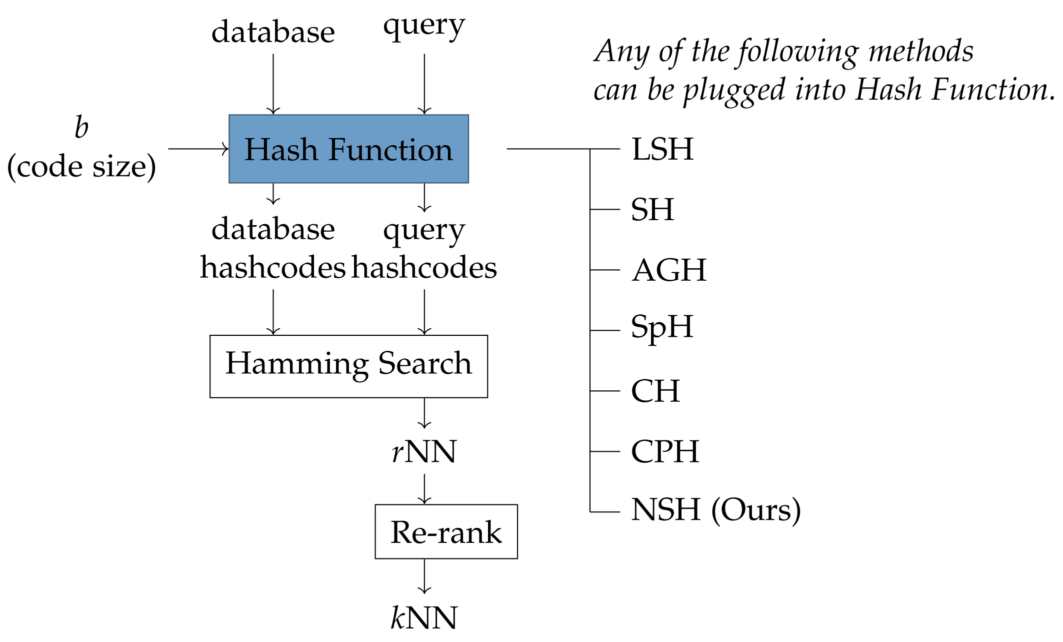


Figure 4.2: The workflow in hashing-based search consists of two main components: Hash Function and Hamming Search. Re-rank is an extra step to boost the search accuracy. This chapter improves the most critical component of this workflow, i.e., Hash Function.

the query item provided at run time into a hashcode. The choice of Hash Function is critical for the accuracy of the *k*NN task. Ideally, the hashcodes should be generated such that the *k*NN set retrieved based on these hashcodes is always identical to the *k*NN set based on the original distances. However, no tractable method is known for achieving this goal; even a simpler problem is proven to be NP-hard [174]. As a result, hashing-based techniques are only *approximate*; they aim to return as many true *k*NN items as possible. In this chapter, we focus on improving the accuracy of this component (or its efficiency, given a required level of accuracy).

The length of the hashcodes (b) is an important design parameter that must be determined at index time. In general, there is a trade-off between the hashcode length and the search speed. The longer hashcodes tend to capture the original distances more accurately, while they slow down the other runtime component, namely the Hamming Search.

Once the hashcodes that capture the original distance information (to some extent) are generated, Hamming Search is responsible for time-efficient retrieval of the *k*NN items in the Hamming space. The simplest approach would be a linear scan during which the distance between the query's hashcode and the hashcode of every item in the database is computed. Although this simple approach improves over a linear scan over the original data vectors, there have been more efficient algorithms, such as *Multi-Index Hashing* (MIH), developed for exact Hamming Search [130] in sub-linear time complexity. Note that this search speedup is only possible because the hashcodes are (small) binary vectors; the data structure cannot be used to speed up the search for general multi-dimensional data representations.

The last component of the system, Re-rank, is a post-lookup re-ranking step designed for mitigating the negative effects of the hashing step on accuracy. Instead of requesting exactly k data items to Hamming Search, we can request r ($\geq k$) data items. Next, we recompute the similarity of each retrieved item to the query item (in their original representations), then sort and choose the top-k among these r items. Naturally, the larger the value of r is, the more accurate the final kNN items are. However, using a larger r has two drawbacks. First, it needs more time to obtain answers from Hamming Search. Second, the re-ranking process takes more time.

4.2.2 Hash Function Design

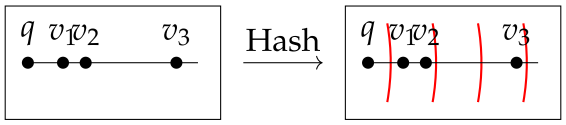
Since the Hash Function choice in Figure 4.2 is independent of the Hamming Search component, the primary objective in designing a good hash function has been finding a hash function that produces high average recalls for a given hashcode length. The Hash Function component is in fact composed of b *bit functions*, each responsible for generating an individual bit of the overall hashcode. Next, we define the role of these bit functions more formally.

Definition 3. (Bit Function) A function h that takes a data item v and produces $h(v) \in \{-1, 1\}$ is called a bit function. Here, v can be a novel query item or any of the existing items in the database. The value of the bit function, $h(v)$, is called a *hash bit*.

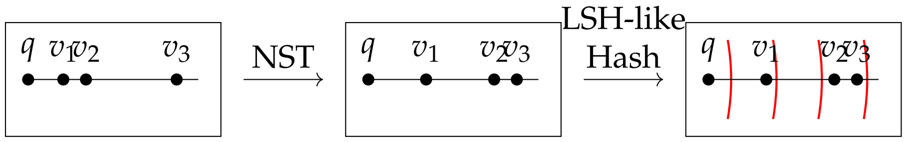
Note that in reality binary bits are stored in their $\{0, 1\}$ representations. However, using signed bits $\{-1, 1\}$ greatly simplifies our mathematical expressions. Thus, we will use signed binary bits throughout the chapter.

Definition 4. (Hash Function) A hash function $\mathbf{h}$ is a series of b bit functions $(h_1, \dots, h_b)$. The hash bits produced by h_1 through h_b are concatenated together to form a *hashcode* of length b.

We consider a hashcode of v as a b-dimensional vector whose elements are either of $\{-1, 1\}$. A natural distance measure between two hashcodes is to count the number of positions that have different hash bits, known as the Hamming distance. As mentioned



(a) Regular Hashing



(b) Hashing with NST

Figure 4.3: The motivation behind using *Neighbor-Sensitive Transformation* (NST) before hashing: applying NST to data items makes the same hashing algorithm place more separators between nearby items (v_1 and v_2), and place fewer separators between distant items (v_2 and v_3).

in Section 5.1, we denote the Hamming distance between data items v_i and v_j by $\|\mathbf{h}(v_i) - \mathbf{h}(v_j)\|_H$.

Finally, we formally state the *locality-sensitive* property [37, 45], which is a widely accepted principle for designing hash functions. Let q be a query, and v_i, v_j be two arbitrary data items. We say that a bit function h satisfies the locality-sensitive property, if

$$\|q - v_i\| < \|q - v_j\| \Rightarrow \Pr(h(q) \neq h(v_i)) < \Pr(h(q) \neq h(v_j)). \tag{4.1}$$

where $\Pr(\cdot)$ denotes the probability.

Datar et al. [45] showed that assigning hash bits based on their relative locations with respect to a set of randomly-drawn hyperplanes satisfies the locality-sensitive property. That is, for a b-bit hashcode, b independent hyperplanes are drawn from a normal distribution to compose a hash function. Using an unlimited number of hash bits using this approach could perfectly capture the original distances. However, many recent algorithms have shown that this simple approach does not achieve high kNN accuracy when the hashcodes need to be short [69, 83, 84, 109, 174].

4.3 Neighbor-Sensitive Hashing

This section describes our main contribution, *Neighbor-Sensitive Hashing* (NSH). First, Section 4.3.1 formally verifies our intuition introduced in Section 5.1: using more separators for nearby data items allows for more accurate distinction of kNN items. As depicted in Figure 4.3, NSH is the combination of a hashing algorithm and our proposed Neighbor-Sensitive Transformation (NST). Section 4.3.2 lays out a set of abstract mathematical properties for NST, and Section 4.3.3 presents a concrete example of NST that satisfies those properties. Lastly, Section 4.3.4 describes our final algorithm (NSH) that uses the proposed NST as a critical component.

4.3.1 Formal Verification of Our Claim

In this section, we formally verify our original claim: using more separators between data items leads to a more successful ordering of data items based on their hashcodes. First, let us formalize the intuitive notions of "having more separators" and "correct ordering based on hashcodes":

- Let q be a query point, v_1, v_2 be two data items where $\|q - v_1\| < \|q - v_2\|$, and $\mathbf{h}$ be a hash function that assigns hashcodes to these items. Then, having more separators between v_1 and v_2 means a larger gap in terms of their Hamming distance, namely $\|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H - \|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H$ will be larger.
- For v_1 and v_2 to be correctly ordered in terms of their distance to q, v_1's hashcode must be closer to q's hashcode compared to v_2's hashcode. In other words, $\|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H - \|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H$ must be a positive value.

In the rest of this chapter, without loss of generality, we assume that the coordinates of all data items are normalized appropriately, so that the largest distance between pairs of items is one. Next, we define the class of hash functions we will use.

Definition 5. (LSH-like Hashing) We call a bit function h *LSH-like*, if the probability of two items v_i and v_j being assigned to different hash bits is linearly proportional to their distance, namely $\Pr(h(v_i) \neq h(v_j)) = c \cdot \|v_i - v_j\|$ for some constant c. We call a hash function $\mathbf{h}$ LSH-like if all its bit functions are LSH-like.

Note that not all existing hashing functions are LSH-like. However, there are several popular hashing algorithms that belong to this class, such as LSH for Euclidean distance [45]. With these notions, we can now formally state our claim.

Theorem 4.1. Let q be a query, and v_1 and v_2 two data items. Also, let $\mathbf{h}$ be an LSH-like hash function consisting of b *independent* bit functions $h_1, \ldots, h_b$. Then, the following relationship holds for all v_1 and v_2 satisfying $0.146 < \|q - v_1\| < \|q - v_2\|$: A larger value of $E\|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H - E\|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H$ implies a larger value of $\Pr(\|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H < \|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H)$, i.e., the probability of successful ordering of v_1 and v_2 based on their hashcodes.

Proof. Let us denote the probability that an individual bit function h assigns q and v_1 into *different* hash bits is p_1, and the probability that assigns q and v_2 into different hash bits is p_2. A reasonable bit function satisfies $p_1 < p_2 \leq 0.5$. Note that $E\|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H - E\|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H = b \cdot (p_2 - p_1)$. There are two cases in which the above expression increases: first, p_1 becomes smaller, and second, p_2 becomes larger. Let us start with the first case.

To compute the probability distribution of the difference of hamming distances, $p(\|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H - \|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H)$, we take a look at the distribution of $\|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H$. Since b number of bit functions that compose $\mathbf{h}$ are independent of one another, $\|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H$ follows the binomial distribution with mean bp_1 and variance $bp_1(1-p_1)$. Similarly, $\|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H$ follows the binomial distribution with mean bp_2 and variance $bp_2(1-p_2)$. Exploiting the fact that binomial distributions can be closely approximated by the normal distributions with the same mean and the variance, and that the difference between two normal distributions follows another normal distribution, we can state the following:

$$\begin{aligned} &p(\|\mathbf{h}(q) - \mathbf{h}(v_1)\|_H < \|\mathbf{h}(q) - \mathbf{h}(v_2)\|_H) \\ &\approx \int_0^\infty \mathcal{N}\left(bp_2 - bp_1, bp_2(1-p_2) + bp_1(1-p_1)\right)\, dx \end{aligned}$$

where $\mathcal{N}$ denotes the probability distribution function of a normal distribution. Note that the above quantity is a function of two values: mean and standard deviation. Due to the shape of a normal distribution, higher mean and smaller standard deviation results in a higher chance of successful ordering of v_2 and v_1 based on their hashcodes. If p_1 decreases, the mean of the above normal distribution increases, and the standard deviation of the distribution decreases. Therefore, the quantity of our interest increases. (End of the first case)

Next, let us discuss the case where p_2 increases. This case asks for more careful analysis because the standard deviation of the normal distribution increases. Recall that the area computed by the integration is a function of the mean and the standard deviation of the normal distribution; thus, *if the mean increases faster than the standard deviation, the quantity of our interest still increases.* Let us compute the condition that the mean increases faster. Since we are dealing with the case in which p_2 increases,

$$\frac{\partial(bp_2 - bp_1)}{\partial p_2} = b \tag{4.2}$$

must be larger than

$$\frac{\partial \sqrt{bp_2(1-p_2)+bp_1(1-p_1)}}{\partial p_2} = b \cdot \frac{1-2p_2}{2\sqrt{p_2 - p_2^2}} \tag{4.3}$$

The condition for this is $p_2 > 0.146$. Then, what is the fraction of the data items that do *not* satisfy this condition? For this, a pair of data items should be assigned to the different hash bits with the probability less than 0.146. For a single dimensional case, the fraction of such area is 0.146/0.5. For a 10-dimensional case, the fraction of such area is $(0.146/0.5)^{10} = 4.5 \cdot 10^{-6}$. Because the dimensions of the data items we are interested in are usually over 100, the fraction of the data items that do not satisfy this condition is very small. (End of the second case) □

This theorem implies that having more separators between two data items helps with their successful ordering using their hashcodes. Since the total number of separators is a fixed budget b, we need to borrow some of the separators that would otherwise be used for distinguishing distant (or non-kNN) items. The following sections describe how this theorem can be used for designing such a hash function.

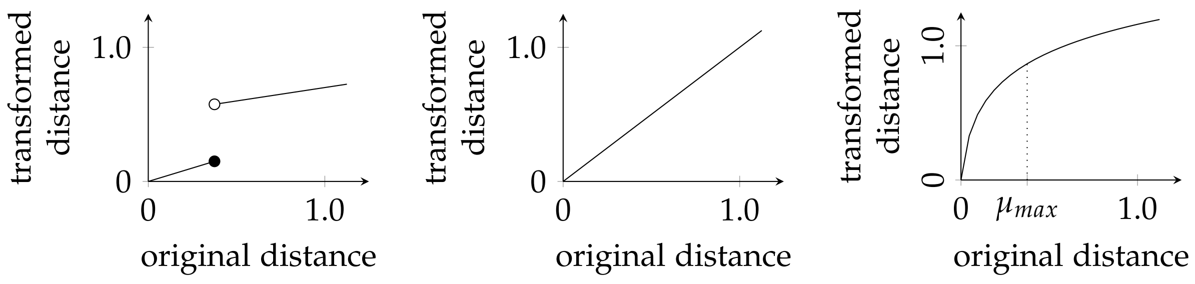
4.3.2 Neighbor-Sensitive Transformation

As shown in Figure 4.3, the main idea of our approach is that combining our *Neighbor-Sensitive Transformation* (NST) with an LSH-like hash function produces a new hash function that is highly effective in distinguishing nearby items. In this section, we first define NST, and then formally state our claim as a theorem.

Definition 6. (Neighbor-Sensitive Transformation (NST)) Let q be a query. A coordinate-transforming function f is a q-neighbor-sensitive transformation for a given distance range (η_{min}, η_{max}), or simply a q-(η_{min}, η_{max})-sensitive transformation, if it satisfies the following three properties:

1. Continuity: f must be continuous.[4]
2. Monotonicity: For all constants t_i and t_j where $t_i \le t_j$, f must satisfy $E(\|f(q) - f(v_i)\|) < E(\|f(q) - f(v_j)\|)$, where the expectations are computed over data items v_i and v_j chosen uniformly at random among items whose distances to q are t_i and t_j, respectively.

[4]This condition is to prevent a pair of similar items in the original space from being mapped to radically different points in the transformed space.



(a) Only Monotonic **(b)** No Larger Gap **(c)** Satisfies All

Figure 4.4: Visual demonstration of NST properties.

3. Larger Gap: For all constants t_i and t_j where $\eta_{min} \leq t_i \leq t_j \leq \eta_{max}$, f must satisfy $E(\|f(q) - f(v_j)\| - \|f(q) - f(v_i)\|) > t_j - t_i$, where the expectation is computed over data items v_i and v_j chosen uniformly at random among items whose distances to q are t_i and t_j, respectively.

The coordinates are re-normalized after the transformation, so that the maximum distance between data items is 1.

To visually explain the properties described above, three example functions are provided in Figure 4.4. Among these three functions, Figure 4.4(c) is the only function that satisfies all three properties — the function in Figure 4.4(a) is neither continuous nor satisfies the Larger Gap property, and the function in Figure 4.4(b) is continuous and monotonic but does not satisfy the Larger Gap property.

The third property of NST (Larger Gap) plays a crucial role in our hashing algorithm. Recall that our approach involves an LSH-like hashing whereby two data items are distinguished in the Hamming space with a probability proportional to their distance. This implies that if we alter the data items to stretch out their pairwise distances, their pairwise Hamming distances are also likely to increase. Thus, such data items become more distinguishable in Hamming space after the transformation.

Thus far, we have defined NST using its three abstract properties. Before presenting a concrete example of a NST, we need to formally state our claim.

Theorem 4.2. Let $\mathbf{h}$ be an LSH-like hash function and f be a q-(η_{min}, η_{max})-sensitive transformation. Then, for all constants t_i and t_j, where $\eta_{min} \leq t_i \leq t_j \leq \eta_{max}$, we have the following:

$$\begin{aligned} &E(\|\mathbf{h}(f(q)) - \mathbf{h}(f(v_j))\|_H - \|\mathbf{h}(f(q)) - \mathbf{h}(f(v_i))\|_H) \\ &> E(\|\mathbf{h}(q) - \mathbf{h}(v_j)\|_H - \|\mathbf{h}(q) - \mathbf{h}(v_i)\|_H) \end{aligned} \tag{4.4}$$

where the expectations are computed over data items v_i and v_j chosen uniformly at random among data items whose distances to q are t_i and t_j, respectively.

Proof. Since $\mathbf{h}$ is LSH-like,

$$\begin{aligned}
E(\|\mathbf{h}(f(q)) - \mathbf{h}(f(v_j))\|_H) &= E_v(E_h(\|\mathbf{h}(f(q))) - \mathbf{h}(f(v_j)\|_H)) \\
&= E_v(c \cdot b \cdot \|f(q) - f(v_j)\|) \\
&= c \cdot b \cdot E_v(\|f(q) - f(v_j)\|)
\end{aligned}$$

where E_h is an expectation over $\mathbf{h}$ and E_v is an expectation over v_j. Similarly,

$$\begin{aligned}
&E(\|\mathbf{h}(f(q)) - \mathbf{h}(f(v_i))\|_H) = c \cdot b \cdot E_v\|f(q) - f(v_i)\| \\
&E(\|\mathbf{h}(q) - \mathbf{h}(v_j)\|_H) = c \cdot b \cdot E_v\|q - v_j\| \\
&E(\|\mathbf{h}(q) - \mathbf{h}(v_i)\|_H) = c \cdot b \cdot E_v\|q - v_i\|
\end{aligned}$$

where E_v is either an expectation over v_i or an expectation over v_j depending on the random variable involved. Due to the third property of NST,

$$E_v\|f(q) - f(v_j)\| - E_v\|f(q) - f(v_i)\| > \|q - v_j\| - \|q - v_i\|$$

Therefore, the relationship we want to show also holds. □

Now that we have established that NST can help with constructing more effective hashcodes, our next task is to find a concrete transformation function that satisfies NST's three properties.

4.3.3 Our Proposed NST

In this section, we first propose a coordinate transformation function for a known query q, and describe its connection to NST (Definition 6). Then, we extend our transformation to handle unknown queries as well. The proposed NST is also a crucial component of our hashing algorithm, which will be presented in the following section.

Definition 7. (Pivoted Transformation) Given a data item p, a pivoted transformation f_p transforms an arbitrary data item v as follows:

$$f_p(v) = \exp\left(-\frac{\|p - v\|^2}{\eta^2}\right) \tag{4.5}$$

where η is a positive constant. We call p the pivot.

For a pivoted transformation $f_p(v)$ to be a q-neighbor-sensitive transformation, we need the distance between p and q to be small enough. The proximity of p and q is determined by the ratio of their distance to the value of η. For example, our lemma below shows that $\|p - q\| < \eta/2$ is a reasonable choice.

To gain a better understanding of the connection between a pivoted transformation and NST, suppose that the pivot p is at the same point as the query q, and that v_1 and v_2 are two data items satisfying $\|q - v_2\| \geq \|q - v_1\|$. We consider two cases: the first is that v_1 and v_2 are close to q so that their distances are less than η, and the second case is that v_1 and v_2 are distant from q so that their distances are much larger than η. In the first case, the distance between v_1 and v_2 after the transformation is $\exp(-\|q - v_1\|^2/\eta^2) - \exp(-\|q - v_2\|^2/\eta^2)$, which tends to be relatively large because the exponential function $\exp(-x^2)$ decrease fast around 1. In the second case, when the data items are far from the query, the value of the exponential function becomes almost zeros, and so does the distance between those data items after the transformation. In other words, the transformation has an effect of stretching out the space near the query while shrinking the space distant from the query.

Next, we establish a connection between a pivoted transformation and NST. First, it is straightforward that a pivoted transformation satisfies *continuity*, since it only uses continuous functions. The second property of NST, monotonicity, is shown by the following lemma.

Lemma 4. A pivoted transformation f_p satisfies the second property of NST, i.e., monotonicity.

Proof. Let q be a query, p be a pivot, and t be an arbitrary positive constant. Also, v is a data item chosen uniformly at random among items whose distance to q is t. In addition, let α denote an angle between $\overrightarrow{qv}$ and $\overrightarrow{qp}$. Since v is chosen uniformly at random, α is a random variable whose probability distribution function is a uniform between 0 and 2π.

To show the monotonicity, it is enough to show the following:

$$\frac{E(|f(q) - f(v)|)}{\partial t} \geq 0 \implies E\left(\frac{\partial |f(q) - f(v)|}{\partial t}\right) \geq 0$$

The interchange of E and the partial derivative is valid since the random variable inside the expectation (v) only depends on α.

To simplify the notations, let $t_q = \|p - q\|$ and $t_v = \|p - v\|$. Then, $t_v^2 = t^2 + t_q^2 - 2t_q t \cos\alpha$ from the law of cosines. Note that t_q is constant while t_v varies depending on

t and α. Therefore,

$$2t_v \frac{\partial t_v}{\partial t} = 2t - 2t_q \cos\alpha, \quad \frac{\partial t_v}{\partial t} = \frac{t - t_q \cos\alpha}{t_v}$$

We divide this proof into two cases: (1) $t \geq 2t_q$ and (2) $t < 2t_q$. For the first case when $t \geq t_q$, we get $t_v \geq t_q$ using the triangular inequality. As a result, $|f(q) - f(v)| = \exp(-t_q^2/\eta^2) - \exp(-t_v^2/\eta^2)$. Therefore,

$$\frac{\partial |f(q) - f(v)|}{\partial t} = \frac{2t_v}{\eta^2} \frac{t - t_q \cos\alpha}{t_v} \exp\left(-\frac{t_v^2}{\eta^2}\right)$$

From $t_v \geq t_q$, we know $2t - 2t_q \cos\alpha \geq t \geq 0$, so

$$E\left(\frac{\partial |f(q) - f(v)|}{\partial t}\right) \geq 0$$

For the second case, when $t < 2t_q$, the sign of $f(q) - f(v)$ depends on the sign of $t_q - t_v$. In other words,

$$\begin{aligned}\frac{\partial |f(q) - f(v)|}{\partial t} &= \frac{f(q) - f(v)}{|f(q) - f(v)|} \frac{2t_v}{\eta^2} \frac{t - t_q \cos\alpha}{t_v} \exp\left(-\frac{t_v^2}{\eta^2}\right) \\ &= \frac{f(q) - f(v)}{|f(q) - f(v)|} \frac{2(t - t_q \cos\alpha)}{\eta^2} \exp\left(-\frac{t^2 + t_q^2 - 2t_q t \cos\alpha}{\eta^2}\right)\end{aligned}$$

We further simplify the above expression by substituting $l_1\eta$ and $l_2\eta$ for t and t_q, respectively, and we treat the expression as a function of l_1 and l_2. Then, we obtain

$$g(l_1, l_2) = \frac{1}{2\pi} \int_0^{2\pi} \frac{f(q) - f(v)}{|f(q) - f(v)|} \frac{2(l_1 - l_2 \cos\alpha)}{\eta} \exp(-l_1^2 + l_2^2 - 2l_1 l_2 \cos\alpha)\, d\alpha$$

Unfortunately, showing that $g(l_1, l_2) > 0$ for all l_1 and l_2 such that $0 < l_2 < 0.5$ and $0 < l_1 << 1$ analytically is difficult because a closed-form solution of the above integration does not exist. However, we can obtain high confidence from numerical analysis due to the following reasons:

1. $g(l_1, l_2)$ is a continuous function of l_1 and l_2.
2. The function of the form $x \exp(x)$ does not fluctuate fast and is can be approximated well by piece-wise linear functions.

Therefore, if we pick four closely located points in the space of l_1 and l_2 and the values of $g(l_1, l_2)$ at all those four points are positive, we obtain high confidence that the function values of the area enclosed by those four points will be positive as well.

For this, we generated 1,000,000 pairs of (l_1, l_2) where l_1 and l_2 are evenly spaced between 0 and 1 and between 0 and 0.5, respectively. Next, we computed the value of the function $g(l_1, l_2)$ for all 1,000,000 pairs. In the end, we confirmed that every pair we generated are positive. This result implies that $g(l_1, l_2) > 0$ for all l_1 and l_2 such that $0 < l_2 < 0.5$ and $0 < l_1 < 2l_2 < 1$. □

The next lemma is regarding the third property of NST, namely a Larger Gap.

Lemma 5. A pivoted transformation f_p with $\|p - q\| < \eta/2$ and $\eta < 0.2$ satisfies the third property of NST, i.e., Larger Gap, for $(\eta_{min}, \eta_{max}) = (0.13\eta, 1.6\eta)$. That is, f_p is a q-$(0.13\eta, 1.6\eta)$-sensitive transformation.[5]

Proof. Let q be a query, p be a pivot, and t be an arbitrary positive constant. Also, v is a data item chosen uniformly at random among items whose distance to q is t. In addition, let α denote an angle between $\overrightarrow{qv}$ and $\overrightarrow{qp}$. Since v is chosen uniformly at random, α is a random variable whose probability distribution function is a uniform between 0 and 2π.

To show the monotonicity, it is enough to show the following:

$$\frac{E(|f(q) - f(v)|)}{\partial t} \geq 1 \implies E\left(\frac{\partial |f(q) - f(v)|}{\partial t}\right) \geq 1$$

for $t \in (0.13\eta, 1.6\eta)$. The interchange of E and the partial derivative is valid since the random variable inside the expectation (v) only depends on α.

To simplify the notations, let $t_q = \|p - q\|$ and $t_v = \|p - v\|$. Then, $t_v^2 = t^2 + t_q^2 - 2t_q t \cos\alpha$ from the law of cosines. Note that t_q is constant while t_v varies depending on t and α. Therefore,

$$2t_v \frac{\partial t_v}{\partial t} = 2t - 2t_q \cos\alpha, \quad \frac{\partial t_v}{\partial t} = \frac{t - t_q \cos\alpha}{t_v}$$

and

$$\begin{aligned} \frac{\partial |f(q) - f(v)|}{\partial t} &= \frac{f(q) - f(v)}{|f(q) - f(v)|} \frac{2t_v}{\eta^2} \frac{t - t_q \cos\alpha}{t_v} \exp\left(-\frac{t_v^2}{\eta^2}\right) \\ &= \frac{f(q) - f(v)}{|f(q) - f(v)|} \frac{2(t - t_q \cos\alpha)}{\eta^2} \exp\left(-\frac{t^2 + t_q^2 - 2t_q t \cos\alpha}{\eta^2}\right) \end{aligned}$$

[5]When working with *non-normalized distances*, η should be smaller than $0.2 \cdot t_{max}$, where t_{max} is the maximum distance between data items.

We substitute $l_1\eta$ and $l_2\eta$ for t and t_q, respectively, and consider the above expression as a function of l_1 and l_2. Then, we should show that

$$g(l_1, l_2) = \frac{1}{2\pi} \int_0^{2\pi} \frac{f(q) - f(v)}{|f(q) - f(v)|} \frac{2(l_1 - l_2 \cos\alpha)}{\eta} \exp(-l_1^2 + l_2^2 - 2 l_1 l_2 \cos\alpha)\, d\alpha \geq 1$$

Since $\eta < 0.2$, it is enough to show that

$$g'(l_1, l_2) = \frac{1}{2\pi} \int_0^{2\pi} \frac{f(q) - f(v)}{|f(q) - f(v)|} 2(l_1 - l_2 \cos\alpha) \exp(-l_1^2 + l_2^2 - 2 l_1 l_2 \cos\alpha)\, d\alpha \geq 0.2$$

for all l_1 and l_2 such that $0 < l_2 < 0.5$ and $0.13 < l_1 < 1.6$. Similar to Lemma 4, analytically computing the above integration is not easy because a closed-form solution of the above integration does not exist. Thus, to numerically verify this lemma, we generated 1,000,000 pairs of (l_1, l_2) where l_1 and l_2 are evenly spaced between 0.13 and 1.6 and between 0 and 0.5, respectively. Next, we computed the value of $g'(l_1, l_2)$ for all pairs. In the end, we confirmed that every pair we generated is not smaller than 0.204. $\square$

A q-$(0.13\eta, 1.6\eta)$-sensitive transformation implies that our intended effect may not be achieved for those data items whose distances to q are smaller than 0.13η. Fortunately, a simple case study shows that the number of such data items is negligibly small: consider 100 million data points that are uniformly distributed in a 10-dimensional space; then, the number of data items that fall within the distance $0.13\,\eta$ (or 0.026) from q will be $100 \cdot 10^6 \cdot 0.026^{10} = 1.4 \times 10^{-8}$. Considering that users are typically interested in a relatively small number of results from their search engines, say the top 1–1000 items, we see that this condition can cover most of the practical cases.

Handing Novel Queries— Thus far, we have described our NST for a known query q. However, we also need to handle queries that are not known *a priori*. Note that, from Lemma 5, we know that NST properties hold for all queries that are within a $\eta/2$ distance from a pivot. Handling queries that are extremely far apart from *all* data items in the database will therefore be difficult. However, assuming that novel queries will be relatively close to at least one of the data items, we can handle such queries by selecting multiple pivots that can collectively cover the existing data items. Based on this observation, we propose the following transformation to handle novel queries.

Definition 8. (Multi-Pivoted Transformation) Let f_p be a pivoted coordinate transformation in Definition 7 using a pivot p. Our extended version to handle novel queries is as follows. Choose m pivots $\{p_1, \ldots, p_m\}$, and compute the below to obtain a multidimensional representation of a data item v:

$$f(v) = (f_{p_1}(v), \ldots, f_{p_m}(v)) \tag{4.6}$$

To understand how a multi-pivoted transformation works for novel queries, assume for the moment that there is at least one pivot p_i that is close enough to a novel query q. Then, this pivot works in the same way as in a single pivoted transformation: it stretches out the distances between this novel query and other data items around it. As a result, when combined with an LSH-like hash function, more separators are used to distinguish q and its nearby items. On the other hand, from the perspective of other (far-apart) pivots, the distances between the q and its nearby items tend to be very small after the transformation, due to the exponential function used in the pivoted transformation. Consequently, those far-apart pivots are effectively ignored by a multi-pivoted transformation when computing the distance of q and its neighbors. This effect of the multi-pivoted transformation is examined empirically in Section 4.4.2. However, one question remains: how can we ensure that there will be at least one nearby pivot for every novel query?

Parameter η— To ensure that there is at least one pivot close enough to each novel query, we use the assumption that each novel query is close to at least one data item in the database. Then, it will suffice to select pivots in such a way that every data item in the database is close to at least one pivot. Specifically, assume that m pivots are chosen by one of the algorithms presented in the next section (Section 4.3.4), and let γ denote the average distance between a pivot and its closest neighbor pivot. Then, to ensure that any data item is within a $\eta/2$ distance from its closest pivot, we should set η to a value larger than γ. This is because the maximum distance between data items and their respective closest pivots will be larger than $\gamma/2$. Our numerical study in Section 4.4.2 shows that, with our choice of $\eta = 1.9\,\gamma$, most of the novel queries fall within a $\eta/2$ distance from their closest pivot. We also show, in Section 4.4.6, that the search accuracy does not change much when η is larger than γ.

For a multi-pivoted transformation, we also need to determine (1) the number of pivots (m) and (2) a strategy for selecting m pivots. We discuss these two issues in Section 4.3.4 after presenting the technical details of our algorithm.

4.3.4 Our NSH Algorithm

This section describes our algorithm, *Neighbor-Sensitive Hashing* (NSH). Besides NST, another ingredient for our hashing algorithm is enforcing the *even distribution* of data items in Hamming space, which is a widely-adopted heuristic. Let h_i^* represent a column vector $(h_i(v_1), \ldots, h_i(v_N))^T$ where $v_1, \ldots, v_N$ are the data items in a database. In other words, h_i^* is a column vector of length N that consists of all i-th bits collected from the generated hashcodes. Then, the technical description of the even distribution of the data points in the Hamming space is as follows:

$$(h_i^*)^T \mathbb{1} = 0 \quad \forall i = 1, \ldots, b \tag{4.7}$$

$$(h_i^*)^T h_j^* = 0 \quad \forall i, j = 1, \ldots, b \text{ and } i \neq j \tag{4.8}$$

The first expression induces that the hash bits are *turned on* with 50% chance. The second expression induces that two hash bits in different positions are uncorrelated so that they have different hash bits in different positions. The second condition also means that the conditional probability that a data item receives 1 for i-th hash bit is independent of the probability that the data item receives 1 for j-th hash bit if $i \neq j$.

The primary objective NSH is to generate a hash function using NST, while best ensuring the above requirement for the data items that reside in the database. First, if we consider a data item v as a d-dimensional column vector, the hash bits are determined by NSH in the following way: $h_i = \text{sign}(w_i^T f(v) + c_i)$, where f is a multi-pivoted transformation with m pivots, w_i is a m-dimensional vector, and c_i is a scalar value. Our main goal in this section is to find the appropriate values for w_i and c_i that can satisfy Equations 4.7 and 4.8. To find these values, NSH performs the following procedure:

1. Choose m pivots.
2. Convert all data items using a multi-pivoted transformation in Definition 8, and obtain N number of m-dimensional transformed items.
3. Generate a vector w_1 and a bias term c_1 using an $(m+1)$-dimensional Gaussian distribution.
4. Adjust the vector w_1 and the bias term c_1 so that the resulting hash bits satisfy Equation 4.7.
5. For each $i = 2$ to b (hashcode length),
    (a) Generate a vector w_i and a bias term c_i from an $(m+1)$-dimensional Gaussian distribution.

(b) Adjust w_i and the bias term c_i so that the resulting hash bits h_i^* satisfy Equations 4.7 and 4.8 with respect to the already generated hash bits h_j^* for $j = 1, \ldots, i-1$.

6. Collect w_i and c_i for $i = 1, \ldots, b$, which compose our hash function of length b.

A natural question is how to adjust the random vectors w_i and compute the bias terms so that the hash bits follow Equations 4.7 and 4.8. For this purpose, we maintain another series of $(m+1)$-dimensional vectors z_j where $j = 1, \ldots, b$. When we generate w_i, the set of vectors z_j for $j = 1, \ldots i$ work as a basis to which w_i must be orthogonal.[6] From now on, we think w_i for $i = 1, \ldots, b$ is $(m+1)$-dimensional vectors including the bias term in its last element. Let F denote a N-by-$(m+1)$ *design matrix*, for which the rows are the transformed data items $(f(v_1), \ldots, f(v_N))$ and the number of the columns is the number of the pivots plus one (the last column is one-padded to be multiplied with the bias component of w_i). Then the collection of i-th hash bits can be expressed compactly as follows: $h_i^* = \text{sign}(Fw_i)$.

When we compute the coefficient w_1 for the first bit function, h_1^* must be orthogonal to $\mathbb{1}$ according to Equation 4.7. As a result, when generating w_1 for the first hash bits, we aim to satisfy the following expression: $\text{sign}(Fw_1)^T \mathbb{1} = 0$. We relax this expression for efficient computation as follows: $w_1^T F^T \mathbb{1} = 0$. From this expression, we can easily see that z_1 can be set to $F^T \mathbb{1} / \text{norm}(F^T \mathbb{1})$, then w_1 is obtained by first generating a random vector l and subtracting the inner product of l and z_1 from l

When we compute the coefficient vector w_2 for the second bit function, it should satisfy the following two conditions according to Equations 4.7 and 4.8:

$$\text{sign}(Fw_2)^T \mathbb{1} = 0, \qquad \text{sign}(Fw_2)^T h_1^* = 0.$$

For computational efficiency, these conditions are relaxed as:

$$w_2^T F^T \mathbb{1} = 0, \qquad w_2^T F^T h_1^* = 0.$$

We can simply ignore the first requirement among the two because z_1 already holds the necessary information. For the second requirement, we set z_2 to $F^T h_1^* - (F^T h_1^*)^T z_1$ and normalize it, which is the component of $F^T h_1^*$ that is orthogonal to z_1. With those two vectors of z_1 and z_2, the process of finding w_2 is as straightforward as before: generate a random vector, project the vector onto the subspace spanned by z_1 and z_2, and subtract the projected component from the random vector. Computing other coefficients w_i for

[6]More concretely, w_1 must be orthogonal to z_1, and w_2 must be orthogonal both to z_1 and z_2, and so on.

```
Algorithm 3: Neighbor Sensitive Hashing
  input : $V = \{v_1, \ldots, v_N\}$, $N$ data items
          $b$, code length
          $\eta$, a parameter for coodinate transformation
  output: $W$, a $(m+1)$-by-$b$ coefficient matrix
1 $P \leftarrow m$ pivots
2 $F \leftarrow$ transform($V$, $P$, $\eta$)                    // Definition 8
3 $W \leftarrow [\,]$
4 $Z \leftarrow F^T \mathbb{1} / \text{norm}(F^T \mathbb{1})$
5 for $k = 1$ to $b$ do
6     $w \leftarrow$ random $(m+1)$-by-1 vector
7     $w \leftarrow w - ZZ^T w$
8     $z \leftarrow F^T \, \text{sign}(Fw)$
9     $z \leftarrow z - ZZ^T z$
10    $Z \leftarrow [Z, \; z/\text{norm}(z)]$                 // append as a new column
11 end
12 return W
```

$i = 3, \ldots, b$ can be performed in the same way. Our algorithm is presented in more detail in Algorithm 3.

The resulting time complexity of the process is $O(Nmd + b(mb + Nm))$, which is linear with respect to the database size. We have empirical runtime analysis in Section 4.4.5.

Number of Pivots (m) and Pivot Selection Strategy— Using a large number of pivots helps keep the ratio of η to the maximum distance small, which is one of the conditions for Lemma 5. However, in practice, we observed that increasing the number of pivots beyond b (where b is the length of hashcodes) only marginally improved the search accuracy. This is shown in Figure 4.9(b). On the other hand, the technical conditions in Equations 4.7 and 4.8 and the time complexity analysis above imply important criteria for determining the number of pivots (m):

1. m must be equal to or larger than the hashcode length (b).
2. The smaller the m, the faster the hashing computation.

For these reasons, we recommend $m = c \cdot b$, where c is a small positive integer, e.g., $1, \ldots, 10$. To obtain a value of m that is well tailored to a given dataset, one can additionally employ a standard *cross validation* procedure that is widely used in machine learning literature. For this, we should first partition our original dataset into two, which are called training set and holdout set, respectively. Next, we generate a b-bit hash function with m pivots based on the training set, and test the performance of the generated hash function by using the holdout set as our queries. This procedure is repeated with different values of m, and the value yielding the highest search accuracy on the holdout set is chosen for the actual hashcode generation process.

Once the number of pivots is determined, we need to generate these pivots. We consider three different strategies for this:

1. Uniform strategy: Given the min and max coordinate of existing data items along each dimension, determine the respective coordinates of the pivots by picking m values from that interval uniformly at random.
2. Random strategy: Pick m data items from the database at random, and use them as pivots.
3. k-means strategy: Run the k-means++ algorithm on the existing items, and use the resulting centroids as pivots.

In Section 4.4.6, we study how these pivot selection strategies produce different search accuracies for different query workloads.

Impact of the Data Distribution— Unlike traditional hashing algorithms such as LSH [18, 37, 45, 61], different data distributions lead to different hash functions in our approach. This effect is due to the pivot selection process; once the m pivots are chosen, the remaining steps of our algorithm are agnostic to the data distribution.

The random and k-means strategies tend to choose more pivots in the dense areas. Thus, the locations of the selected pivots are balanced for a balanced dataset, and are skewed for a skewed dataset. In contrast, the uniform strategy is not affected by the skewness of the data distribution, Rather, it is only affected by the range of the data items (i.e., the boundary items).

Our search results are more accurate when there are data items around queries. This is because our algorithm is more effective when there is a pivot close to each query, and we select pivots from areas where data items exist. Thus, when the random or k-means strategy is used, our algorithm is more effective when queries are mostly from the dense areas. When the uniform strategy is used, our algorithm is effective when queries are generated *uniformly* at random within the data items' boundary.

On the other hand, since our algorithm is based on a q-(η_{min}, η_{max})-sensitive transformation, we may not be successful at retrieving all k items when there are fewer than k items within a η_{max} distance of the query. We empirically study this observation in Section 4.4.6.

Finally, note that even for uniformly distributed items, our proposed approach outperforms traditional counterparts for kNN tasks. This is because the core benefit of our approach lies in its greater distinguishability for nearby data items, which is still valid for

Dataset	# Items	Dim	Note
`MNIST` [169]	69,000	784	Bitmap datasets
`80M Tiny` [169]	79,301,017	384	GIST image descriptors
`SIFT` [80]	50,000,000	128	SIFT image descriptors
`LargeUniform`	1,000,000	10	Standard uniform dist.[8]
`SmallUniform`	10,000	10	Standard uniform dist.
`Gaussian`	10,000	10	Standard normal dist.[9]
`LogNormal`	10,000	10	A log-normal dist.[10]
`Island`	10,000	10	`SmallUniform` + two clusters. See Section 4.4.6.

Table 4.1: Dataset Summary. Three real and five synthetic datasets in order. For each dataset, 1,000 data items were held out as queries.

uniform distributions. Our empirical studies in Section 4.4.3 confirm that our technique outperforms not only LSH but also other state-of-the-art learning-based approaches even for uniform datasets.

4.4 Experiments

The empirical studies in this section have two following goals: first, we aim to verify our claim (more hash bits for neighbor items) with numerical analysis, and second, we aim to show the superiority of our algorithm compared to various existing techniques. The results of this section include the following:

1. *Neighbor-Sensitive Transformation* enlarges the distances between close by data items, and the same goal is achieved for the hashcodes generated by *Neighbor-Sensitive Hashing* in terms of their Hamming distance.
2. Our hashing algorithm was robust for all settings we tested and showed superior performance in *k*NN tasks. Specifically, our method achieved the following improvements:
   (a) Up to 15.6% recall improvement[7] for the same hashcode length,
   (b) Up to 22.5% time reduction for the same target recall.

We start to describe our experimental results after stating our evaluation settings.

4.4.1 Setup

Datasets and Existing Methods— For numerical studies and comparative evaluations, we use three real image datasets and five synthetic datasets. A *database* means a collection of data items from which the k most similar items must be identified, and a *query set* means a set of query items we use to test the search performance. As in the general search setting, the query set does not belong to the *database* and is not known in advance; thus, offline computation of kNN is impossible. Table 4.1 summarizes our datasets.

For a comprehensive evaluation, we compared against three well-known approaches and five recent proposals: *Locality Sensitive Hashing* (LSH) [45], *Spectral Hashing* (SH) [174], *Anchor Graph Hashing* (AGH) [113], *Spherical Hashing* (SpH) [69], *Compressed Hashing* (CH) [109], *Complementary Projection Hashing* (CPH) [83], *Data Sensitive Hashing* (DSH) [57], and *Kernelized Supervised Hashing* (KSH) [112]. Section 4.5 describes the motivation behind each approach. Except for CH and LSH, we used the source code provided by the authors. We did our best to follow the parameter settings described in their work. We exclude a few other works that assume different settings for kNN item definitions [75]. For our algorithm, we set the number of pivots (m) to $4b$ and used k-means++ [22] to generate the pivots unless otherwise mentioned. The value of η was set to 1.9 times of the average distance from a pivot to its closest pivot. (Section 4.4.6 provides an empirical analysis of different pivot selection strategies and different values of m and η parameters.) All our time measurements were performed on a machine with AMD Opteron processor (2.8GHz) and 512GB of memory. All hashing algorithms used 10 processors in parallel.

Quality Metric— Recall that Hamming Search is the component responsible for searching in the Hamming space. Once the Hamming Search component returns r candidate items, finding the k most similar items to a query is a straightforward process. Note that if there exists a data item that belongs to the *true* kNN among those r items returned, the data item is always included in the answer set returned by Re-rank. Therefore, a natural way to evaluate the quality of the entire system is to measure the fraction of the data items that belong to the true kNN among the r data items returned by Hamming Search.

[7]Recall improvement is computed as (NSH's recall - competing method's recall).

[8]For each dimension, the standard uniform distribution draws a value between 0 and 1, uniformly at random.

[9]For each dimension, a value is drawn from a Gaussian distribution with a mean value of 0 and a standard deviation of 1.

[10]Log-normal is a popular heavy-tailed distribution. In our experiments, we used $(\mu, \sigma) = (1, 0)$.

In other words, we compute the following quantity:

$$recall(k)@r = \frac{(\text{\# of } \textit{true } k\text{NN in the retrieved})}{k} \times 100. \tag{4.9}$$

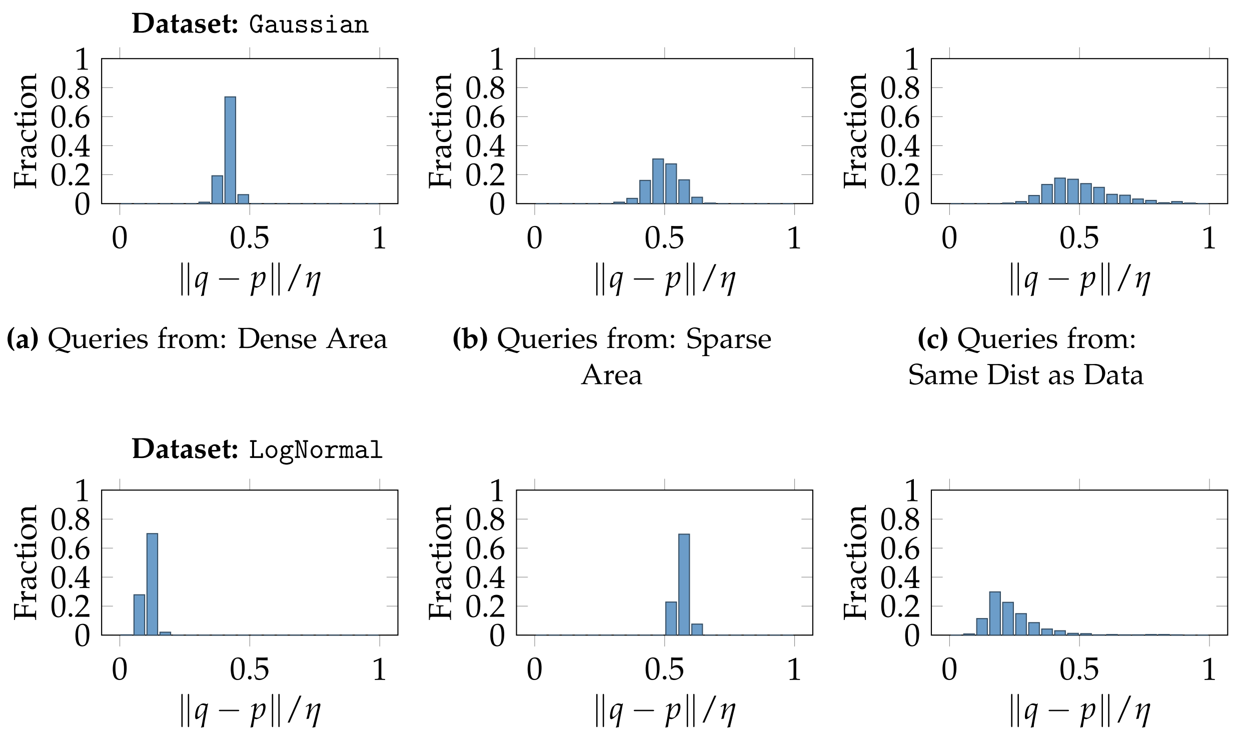
This is one of the most common metrics for evaluating approximate kNN systems [79, 80, 151]. When choosing the r data items with the smallest Hamming distance from a query, it is possible for multiple items to have the same distance at the decision boundary. In such case, a subset of them are chosen randomly. This implies that the recall score of a trivial approach i.e., mapping all the data items to the same hashcodes, cannot be high. Typically, r is chosen as $r = 10k$ or $r = 100k$ to keep the Re-rank speed fast.[11]

Evaluation Methodology— Note that the choice of the Hash Function is independent of the Hamming Search component, and using a different algorithm for Hamming Search can only affect the runtime. Thus, we consider two evaluation settings in this chapter.

1. Hashcode Length and Recall: This setting is to purely evaluate the quality of hashing algorithms without any effects from other components. This evaluation setting is to answer the following question: "what is the best hashing algorithm if the search speed is identical given the same hashcode size?" This evaluation is repeated for different choices of hashcode sizes, because the accuracy of different hashing algorithms can differ greatly based on their hashcode size. For example, it is not uncommon if some methods become less effective as the hashcode size increases.
2. Search Speed and Recall: Another evaluation methodology is to study the tradeoff between search speed and resulting search accuracy. The search time consists of the time required to convert a query into a hashcode and the time to find the rNN data items in the Hamming space. Usually, the time required for hashcode generation is marginal compared to the Hamming Search process.

Another important criteria is the memory requirement for the generated hashcodes. This quantity can be easily inferred from the hashcode size because the size of the bit vectors in memory is identical regardless of the hashing algorithm used. When there are N data items in a database and we generate hashcodes of length b, the amount of memory required to store all hashcodes is $Nb/8$ bytes (since hashcodes are stored as bit vectors). Usually, this quantity is several orders of magnitude smaller than the size of the original database (e.g., 128-dimensional floating point type vectors will take $128 \cdot 4 \cdot N$ bytes), making it possible to keep all the hashcodes in memory.

[11] Larger values of r (e.g., close to the total number of items in the database) will improve recall; however, this will also make the search process as slow as the exact kNN.



(a) Queries from: Dense Area **(b)** Queries from: Sparse Area **(c)** Queries from: Same Dist as Data

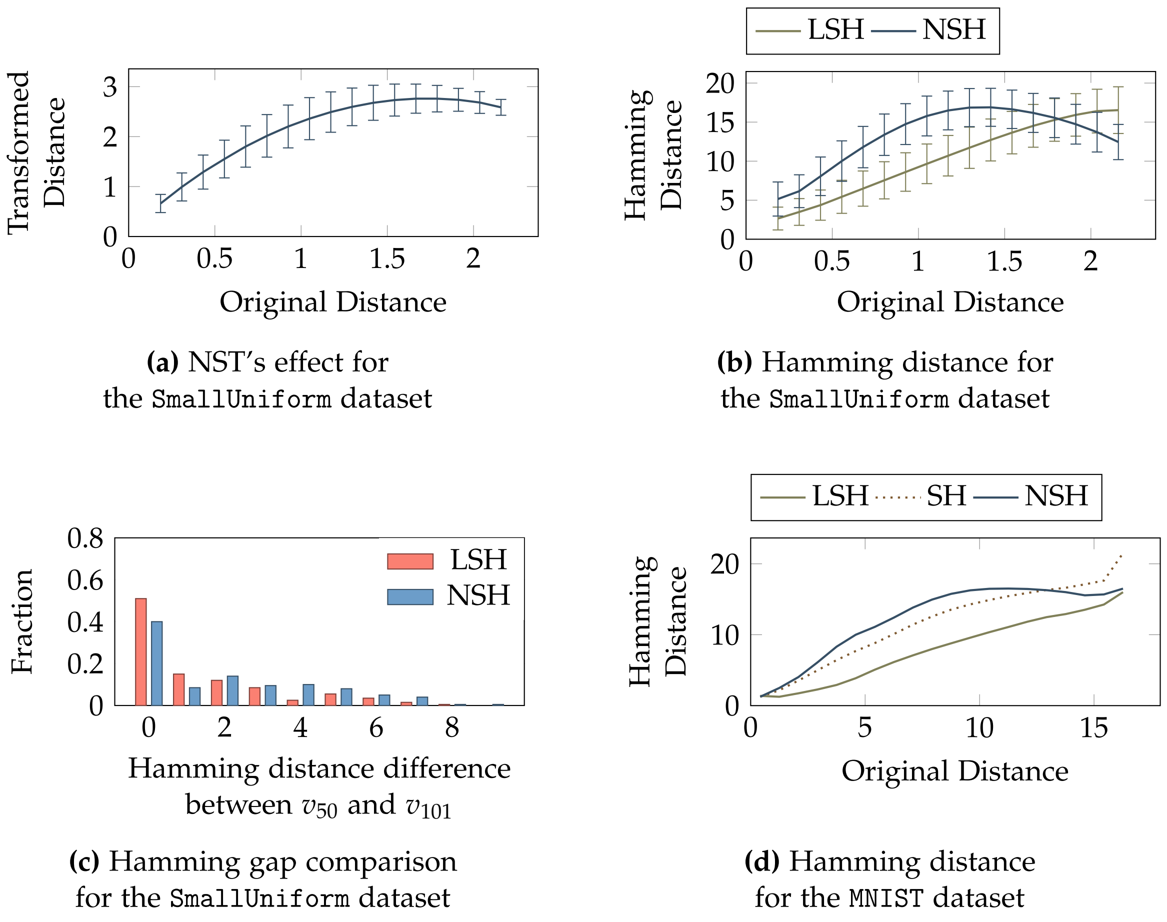
(d) Queries from: Dense Area **(e)** Queries from: Sparse Area **(f)** Queries from: Same Dist as Data

Figure 4.5: The histogram of distances between queries q and their respective closest pivots p divided by η (the parameter from Definition 7).

4.4.2 Validating Our Main Claims

In this section, we numerically verify two important claims we have made: (i) the distance between novel queries and their closest pivot is small, and (ii) NST and NSH achieve their intended goal of placing more separators between closeby items.

First, to study how data items are mapped to pivots, we used `Gaussian` and `LogNormal` as representatives of balanced and skewed datasets, respectively. For each dataset, we considered three different cases: (1) when queries are chosen from the dense area of the dataset, (2) when they are drawn from the sparse area (i.e., outlier items), and (3) when queries come from the same distribution as the original dataset. In each case, we selected 128 pivots using the k-means strategy and measured the distances between queries and their respective closest pivot. Figure 4.5 shows the histogram of these distances. The results show that, for queries from the dense area and from the same distribution as the datasets, the measured distances mostly belong to the range with which NSH can work effectively, i.e., smaller than $\eta/2$. For queries from the sparse area, there were some cases where the measured distances were outside the desired range. This is



(a) NST's effect for the `SmallUniform` dataset

(b) Hamming distance for the `SmallUniform` dataset

(c) Hamming gap comparison for the `SmallUniform` dataset

(d) Hamming distance for the `MNIST` dataset

Figure 4.6: The effects of NST and NSH. Figure (a) shows that NST enlarges the distances among nearby data items. Figure (b) shows that NSH makes nearby data items have larger Hamming distances compared to LSH. Figure (c) shows that there are more separators (hence, a larger Hamming distance gap) between pairs of data items when they are close to queries. Figure (d) shows using a real dataset (`MNIST`) that NSH produces larger Hamming distances between nearby data items compared to SH (a learning-based algorithm) and LSH.

expected since our pivot selection strategy assumes that queries are from an area where existing data items reside. Interestingly, as shown in Section 4.4.6, our final hashing algorithm still provides reasonable performance even for those queries that are drawn from sparse areas.

We also studied whether our proposed mechanisms (NST and NSH) achieve their intended properties, namely increasing the distance gap exclusively for close-by items. Here, we first transformed the data items of the `SmallUniform` dataset using 128 pivots to see how our multi-pivoted transformation (from Definition 8) alters the distances between items. Figure 4.6(a) shows the result. When the original distances between two

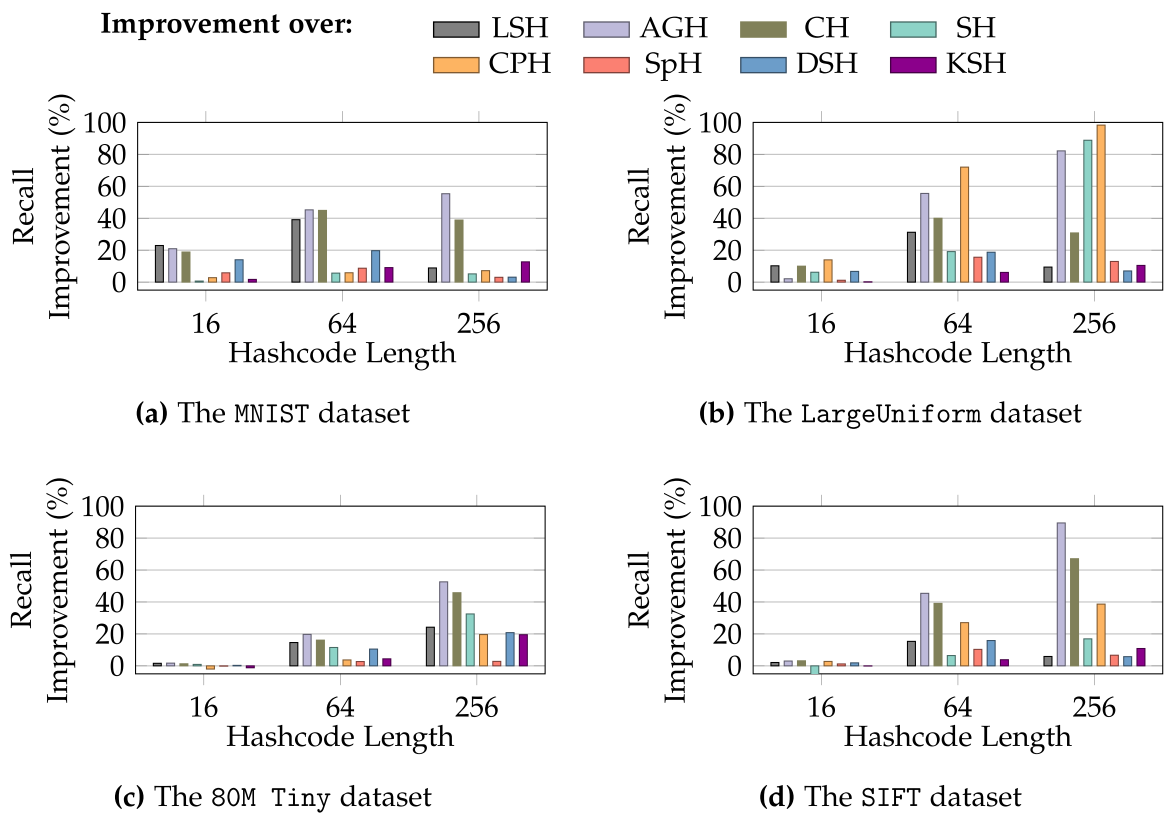


Figure 4.7: Hashcode length and recall improvements. The recall improvement is computed as *(NSH's recall - competing method's recall)*.

items were smaller than the average distance between a query and its *k*-th closest item (e.g., 0.53 for 10-NN, 0.71 for 100-NN, and 0.97 for 1000-NN), their distances were amplified by NST in the transformed space. When we generated 32-bit hashcodes using NSH and LSH, NSH also produced larger Hamming distances (compared to its traditional counterpart, LSH), as reported in Figure 4.6(b). Figure 4.6(c) depicts the same effect by NSH, but using a histogram of the number of separators between the 50th and 101st closest data items to queries. Figure 4.6(d) shows NSH's effectiveness using a real dataset (`MNIST`). Here, NSH again achieved larger Hamming distance gaps than both SH (a learning-based hashing algorithm) and LSH. Note that while the difference between NSH and SH may seem small, it translates to a significant difference in search performance (see Sections 4.4.3 and 4.4.4).

4.4.3 Hashcode Length and Search Accuracy

Recall that the hashcode length (b) is an important design parameter that determines the accuracy of approximate kNN and runtime of Hamming Search. In general, those two factors (search accuracy and runtime) are in a trade-off relationship, i.e., larger hashcodes result in a more accurate but also slower search, and vice versa.

This subsection compares the search accuracies of various hashing algorithms with fixed hashcode lengths. For this experiment, we used the four datasets (`MNIST`, `LargeUniform`, `80M Tiny`, and `SIFT`) and generated different lengths of hashcodes ranging from 16 to 256. Next, we examined how accurately different algorithms capture 10-NN data items for novel queries. For this, we report $recall(10)@100$ for the two relatively small datasets (`MNIST` and `LargeUniform`) and $recall(10)@1000$ for the other two large datasets. We present the experimental results for different choices of k in Section 4.4.7.

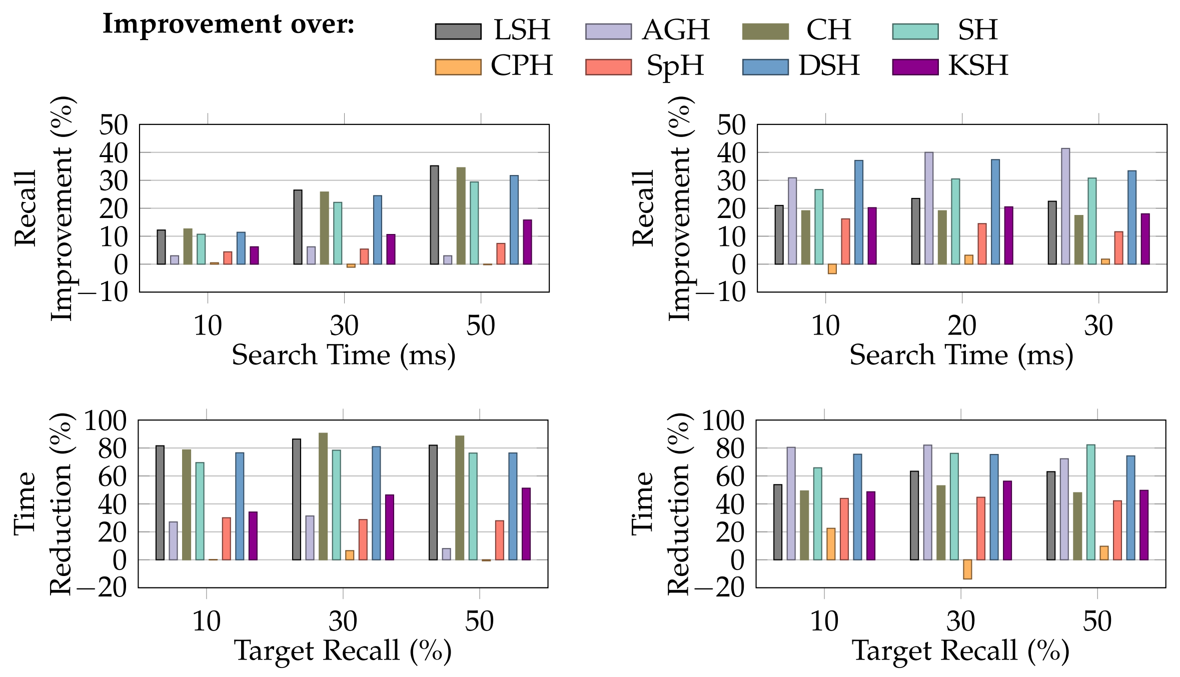
Figure 4.7 shows the results. we report the recall improvements over other competing hashing algorithms. In most cases, the second best methods were SpH and KSH. However, the other recently developed algorithms (such as SH and CPH) worked relatively well too. AGH and CH showed surprisingly bad performance. In all cases, our proposed algorithm showed significant search accuracy gains, showing up to 15.6% improvement of recall over SpH and up to 39.1% over LSH.

4.4.4 Search Time and Search Accuracy

The second setting for performance evaluation is seeing the recall scores by different hashing algorithms when the search time is bounded. For the Hamming Search module, we used *Multi-Index Hashing* (MIH) [130], a recently developed data structure for exact kNN search in Hamming space. MIH has a parameter that determines the number of internal tables, and the search speed varies depending on the parameter setting. We followed a few different options based on the suggestions by its author, and reported the best results for each hashing algorithm.[12]

There are two ways we can improve the search accuracy at the cost of search speed. The first is to increase the hashcode length, and the second is to increase the number of data items returned by Hamming Search (r) and let Re-rank find the k most similar answers. When we tested the first approach, however, we observed that MIH's performance degrades rapidly whenever the hashcode length is over 128, and MIH did not show considerable speed boost compared to a linear scan over hashcodes. For this reason, we used the second approach — increasing the value of r — to adjust the search accuracy and search speed. Then, we collected all 64-bit hashcodes generated by different hashing algorithms, configured MIH to return different number (r) of data items as answers, and measured the recall scores of those answers as well as the time MIH took

[12]We set the number of tables to either 2 or 3.



(a) The `80M Tiny` dataset **(b)** The `SIFT` dataset

Figure 4.8: Search time and recall improvements. The recall improvement is computed as *(NSH's recall - competing method's recall)*. Time reduction is *(competing method's search time - NSH's search time) / (competing method's search time)* $\times$100.

to return them. Note that even if we use the same data structure (MIH) for Hamming Search, systems with different hashing algorithms produce very different results since the hashing mechanism is key in producing high search accuracies.

Figure 4.8 reports recall improvements for a target time bound using two large datasets of `80M Tiny` and `SIFT`. In most cases, NSH showed significant improvements over existing methods. Also, it is impressive that the system with our algorithm achieved as high as 50% average recall for `80M Tiny` within only 49 ms search time requirement. Note that a simple linear scan over the original data items took more than 17 seconds.

4.4.5 Indexing Speed

As we generate hashcodes of different lengths in the above experiments, we also measured the times they took to generate hash functions and to convert the whole database (`80M Tiny`) to hashcodes. The result is summarized in Table 4.2. CPH uses expensive statistical optimization to generate hash functions, so it took a much longer time than other methods. All other methods, including NSH, reported reasonable indexing time.

Method	Hash Gen (sec)		Compression (min)	
	32bit	64bit	32bit	64bit
LSH	0.38	0.29	22	23
SH	28	36	54	154
AGH	786	873	105	95
SpH	397	875	18	23
CH	483	599	265	266
CPH	34,371	63,398	85	105
DSH	3.14	1.48	24	23
KSH	2,028	3,502	24	29
NSH (Ours)	231	284	37	46

Table 4.2: Time requirement for hash function generation and database compression, i.e., converting 79 million data items in a database to hashcodes.

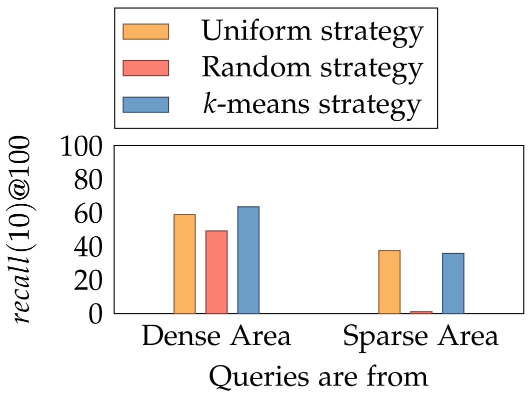
4.4.6 The Effect of Parameters on NSH

This section studies the effect of various parameters on the search accuracy of NSH. The parameters we consider are pivoting (the number of pivots and the selection strategy), the neighborhood parameter (η), the type of query workloads, and the data distribution.

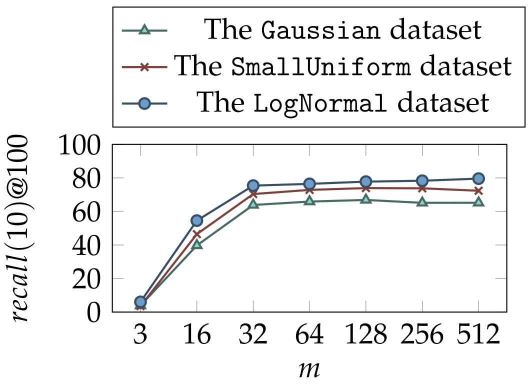
Pivot Selection— As discussed after presenting Definition 8, the goal of choosing pivots is to ensure that the average distance of every data item in the database to its closest pivot is minimized. We studied the three different strategies described in Section 4.3.4: uniform strategy, random strategy, and k-means. For each strategy, we generated 32-bit hashcodes and used the `Gaussian` dataset with two different sets of queries: one set from the dense area (center of the normal distribution) and the other from the sparse area (tail of the distribution). Figure 4.9(a) shows the results. Both uniform and k-means strategies produced almost the same search accuracy, regardless of the query workload. However, the random strategy failed for queries from the spare area. This is because most of the randomly-chosen pivots are naturally from dense areas; thus, the pivots cannot cover the queries from sparse areas. Also, in our experiments, k-means strategy exhibited slightly better performance than the uniform one.

The number of pivots (m) is also important. To empirically study the effect of m on search accuracy, we generated 32-bit hashcodes for three datasets (`Gaussian`, `SmallUniform`, and `LogNormal`) and varied m between 3 and 512. Figure 4.9(b) shows the results. The results suggest that our algorithm is not very sensitive to the value of m, as long as $m \geq b$.

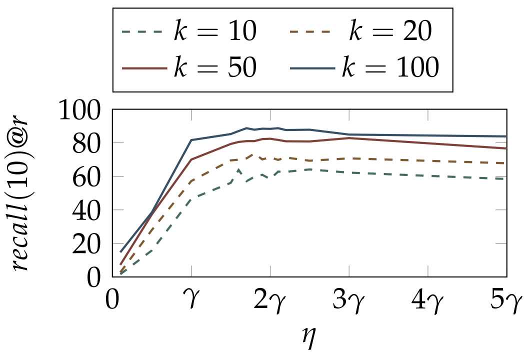
Neighborhood Parameter (η)— The value of η is closely related to the size of the neighborhood for which we want to amplify the Hamming gap. To see η's effect on our



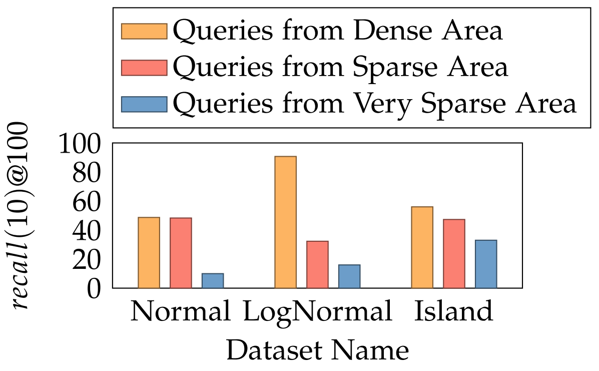
(a) Pivoting Strategy



(b) Numbers of Pivots



(c) Neighborhood Size

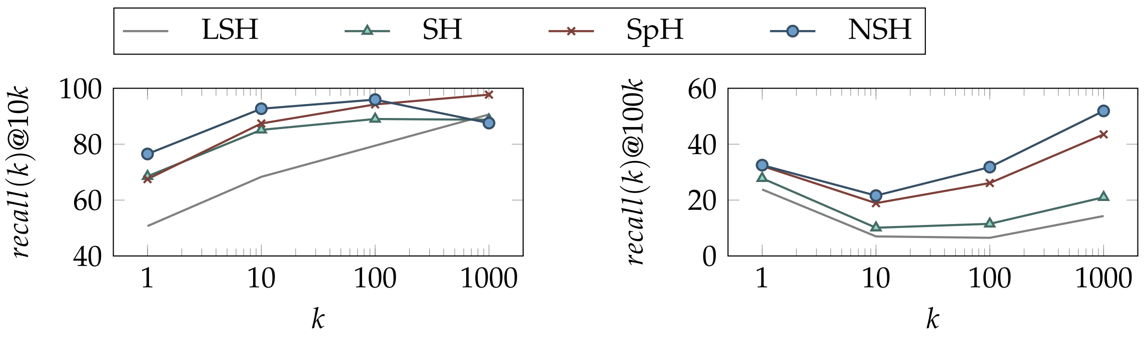


(d) Data distribution

Figure 4.9: We study our method's search performance by varying four important parameters: (a) the pivot selection strategy, (b) the number of pivots, (c) neighborhood parameter η, and (d) the data distribution.

algorithm's performance, we generated 32-bit hashcodes for the `MNIST` dataset and measured the recall while varying η between 0 and 5γ, where γ was the average distance between pairs of closest pivots. The results, plotted in Figure 4.9(c), indicate that our algorithm yields high accuracy when $\eta > \gamma$, and its accuracy curve improves unto $\eta \approx 2\gamma$. Note that this empirical result is consistent with our discussion in Section 4.3.3.

Data Distribution— To study the effect of data distribution, we generated two datasets from standard distributions: `Gaussian` and `LogNormal`. Note that `LogNormal` has a skewed distribution with a heavy tail. In addition, to see how our *k*NN search accuracy is affected when there are fewer than *k* data items around a query, we created another dataset, called `Island`. In `Island`, we added two small clusters to the `SmallUniform` dataset, where each clusters consisted of 3 data items, and they were placed far away (a distance of 1 and 5, respectively) from other data items.



(a) The `MNIST` dataset **(b)** The `80M Tiny` dataset

Figure 4.10: kNN Accuracies with different values of k.

For each dataset, we generated three different queries. For the `Gaussian` dataset, the first query was from its mode.[13] The second and the third queries were twice and three times the standard deviation away from the mode, respectively. The three queries were similarly generated for `LogNormal`. For `Island`, the first query was from the area where the `SmallUniform` dataset resides, and the second and the third queries were from the two small clusters we added. Due to the placements of the two clusters, the third query was much far away from the other data items compared to the second query. For every dataset, we refer to these three queries as 'Q from Dense', 'Q from Sparse', and 'Q from Very Sparse', respectively.

We repeated each experiment 30 times, each time with a different random seed, and reported the average recall scores in Figure 4.9(d). In general, the performance drops were more significant for queries drawn from 'Very Sparse' areas. One reason is the lack of nearby pivots around such queries. The second reason (especially for the `Island` dataset) is that the distance to kNN items were outside the neighborhood size for which NSH can work effectively.

4.4.7 Neighbor Sensitivity

Since our motivation was moving separators (or equivalently, bit functions) to have higher power in distinguishing neighbor items, it is likely that our algorithm loses its power to capture the similarities to the distant items. This potential concern leads to a natural question: up to what value of k does our algorithm have advantage over other algorithms?

[13]A mode is the point at which a probability distribution function takes its maximum.

To answer this question, we varied the value of k while fixing $r = 10 \cdot k$ (Recall r is the number of the items returned by Hamming Search, and Re-rank module returns final k answers) and observed how the search accuracy changed. More concretely, we generated 128-bit and 64-bit hashcodes respectively for `MNIST` and `80M Tiny`, and varied k from 1 to 1,000. See Figure 4.10 for the results.

Interestingly, for `MNIST`, we observed that our algorithm outperformed other methods only up to until $k = 100$ (0.14% of the database). The reason that our method showed superior performance only up to $k = 100$ was that due to the small size of the dataset (only 69K items in the database). When we ran the same experiment with a big dataset (`80M Tiny`), we did not observe the performance decrease until $k = 1000$, and our algorithm consistently outperformed other methods regardless of the choice of k. Considering that the dataset sizes in the real-world are large — the reason of approximate kNN— our algorithm can achieve superior performance in most practical cases.

4.5 Related Work

The growing market for 'Big Data' and interactive response times has created substantial interest in Approximate Query Processing both from academia [14, 47, 134, 187] as well as the commercial world [6, 7, 15]. While these techniques focus on general aggregate queries, this work focuses on approximating kNN queries as an important sub-class of them (see [127] and the references within).

Gionis et al. [61] were the first to apply Locality Sensitive Hashing to approximate search problems. In their work, unary representation of integers were used as hash functions to generate binary hash codes. Later, Charikar [37] proposed to use random hyperplanes as hash functions. These random hyperplanes were drawn from multi-dimensional Gaussian distributions, and generated hashcodes that could retain the locality sensitive property for cosine similarity. Datar et al. [45] proposed a variant of random hyperplane method for the Euclidean distance. Athitsos et al. [24] employed L1-embedding for a similar purpose. Distance-based Hashing [25] generalizes this technique to non-Euclidean distances.

Recent work in this area, however, has started to exploit statistical optimization techniques to learn more intelligent hash functions. These techniques, known as *learning-based* or *data-dependent* hashing, take the distribution of data into account in order to obtain more effective hash functions. A notable approach in this category is Spectral Hashing [174], which motivated others including Binary Reconstructive Embedding [99], Anchor Graph Hashing [113], Random Maximum Margin Hashing [84], Spherical Hash-

Method	Year	Motivation / Intuition
LSH [45]	2004	Random hyperplanes tend to preserve locality
SH [174]	2009	Minimize Hamming distance in proportion to similarities
AGH [113]	2011	Speed up SH by approximation
SpH [69]	2012	Spheres for capturing similarities
CH [109]	2013	Adopt sparse coding theory
CPH [83]	2013	Hash functions should go through sparse areas
DSH [57]	2014	Keep *k*NN items together using Adaptive Boosting

Table 4.3: Several Notable Hashing Algorithms.

ing [69], Compressed Hashing [109], Complementary Projection Hashing [83], and Data Sensitive Hashing [57]. All these methods use different motivations to learn more accurate hashcodes for the *k*NN task. See Table 4.3 for a summary.

There are several techniques developed for efficient indexing and querying of the hashcodes generated by LSH [55, 117, 153, 166, 168, 180]. As explained in Section 4.2.1 these methods belong to the Hamming Search stage; thus, they are orthogonal to our contribution. In our implementation, we employed *Multi-Index Hashing* (MIH) [130], as the state-of-the-art in this area.

Finally, it is important to note that using alternative representations of the original data points for hashing is not a new topic. Different approaches have used different representations to achieve their own motivations. For instance, AGH [113] used one to speed-up SH [174], and CH [109] used one to obtain sparse representations. When we used their representations in place of ours, the resulting algorithm produced a much worse performance. Finally, in machine learning, non-linear transformations have also been used for learning distance metrics for *k*NN classifiers [93, 124].

4.6 Summary

In this chapter, we have presented *Neighbor-Sensitive Hashing*, a algorithm that improves approximate *k*NN search based on an unconventional observation that magnifying the Hamming distances among neighbors helps in their accurate retrieval. We have formally proven the effectiveness of this novel strategy. We have also shown empirically that NSH yields better recall than its state-of-the-art counterparts given the same number of hash bits. NSH is a “drop-in replacement” for existing hashing methods. As a result, any application that chooses NSH can either enjoy an improved search quality, or trade NSH’s recall advantage for a significant speed-up (i.e., reduced time and memory footprint by using shorter hashcodes).

This chapter demonstrated the possibility of speeding up AQP by building task-aware synopses. The *k*NN algorithm, which this chapter has focused on, is an essential building block for many data mining algorithms, such as *k*-nearest neighbor classifiers and collaborative filtering. We believe our contribution can improve the performance of many advanced data analytics tasks that rely on retrieving similar items. In the following chapter, we present another technique that speeds up AQP by building task-aware synopses.

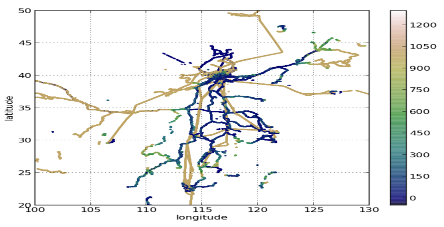
Chapter 5

High-quality Approximate Visualizations by Learning from Data

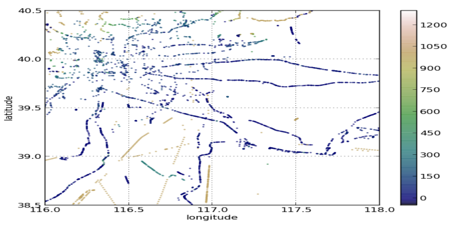
In this chapter, we present our second technique that speeds up AQP by building task-ware synopses. This technique speeds up approximate visualizations by choosing a subset of data items wisely. Similar to the previous techniques we have presented in this dissertation, the technique in this section also produces higher-quality results (here, visualizations) given fixed time-bounds, or requires shorter query processing time for meeting certain target quality.

5.1 Motivation

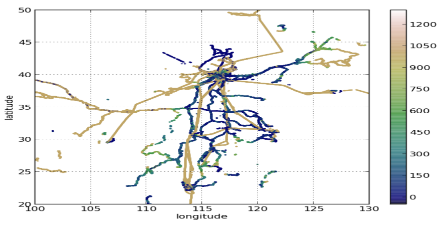
Data scientists frequently rely on visualizations for analyzing data and gleaning insight. For productive data exploration, analysts should be able to produce *ad hoc* visualizations in interactive time (a well-established goal in the visualization and human-computer interaction (HCI) community [28, 42, 43, 51, 52, 67, 110, 115, 142, 175]). However, with the rise of big data and the growing number of databases with millions or even billions of records, generating even simple visualizations can take a considerable amount of time. For example, as reported in Figure 5.2, we found that the industry standard Tableau visualization system takes over 4 minutes on a high-end server to generate a scatterplot for a 50M-tuple dataset that is already resident in memory. (see Section 5.6.1 for experimental details.) On the other hand, HCI researchers have found that visualizations must be generated in 500ms to 2 seconds in order for users to stay engaged and view the system as interactive [114, 123, 157]. Unfortunately, dataset sizes are already growing faster than Moore's Law [165] (the rate at which our hardware is speculated to improve), so technology trends will likely exacerbate rather than alleviate the problem.



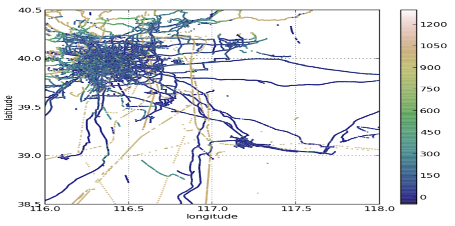
(a) Stratified Sampling (overview)



(b) Stratified Sampling (zoom-in)



(c) VAS (overview)



(d) VAS (zoom-in)

Figure 5.1: Samples generated by fined-grained stratified sampling and our approach respectively. When the entire range is visualized, both methods seem to offer the visualization of the same quality. However, when zoomed-in views were requested, only our approach retained important structures of the database.

This chapter addresses the problem of interactive visualization in the case of *scatterplots* and *map plots*. Scatterplots are a well-known visualization technique that represent database records using dots in a 2D coordinate system. For example, an engineer may investigate the relationship between time-of-day and server-latency by processing a database of Web server logs, setting time-of-day as the scatterplot's X-axis and the server-latency as its Y-axis. Map plots display geographically-tied values on a 2D plane. Figure 5.1(a) is an example of a map plot, visualizing a GPS dataset from OpenStreetMap project with 2B data points, each consisting of a latitude, longitude, and altitude triplet (altitude encoded with color).

One approach for reducing the time involved in visualization production is via *data reduction* [29]. Reducing the dataset reduces the amount of work for the visualization system (by reducing the CPU, I/O, and rendering times) but at a potential cost to the quality of output visualization. Effective data reduction will shrink the dataset as much as possible while still producing an output that preserves all important information of the original dataset. Sampling is a popular database method for reducing the amount of data to be processed, often in the context of approximate query processing [13,15,27,38,

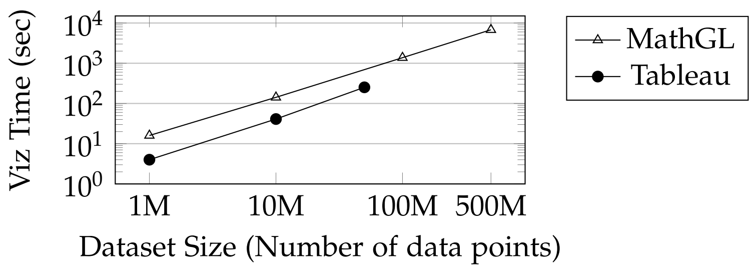


Figure 5.2: The latency for generating scatter plot visualizations using Tableau and MathGL (a library for scientific graphics).

68,81,125,131,187]. While uniform (random) sampling and stratified sampling are two of the most common and effective approaches in approximate query processing [127], they are not well-suited for generating scatter and map plots: they can both fail to capture important features of the data if they are sparsely represented [115].

Figure 5.1 depicts an example using the Geolife dataset [189]. This dataset contains GPS-recorded locations visited by the people living in and around Beijing. In this example, we visualized 100K datapoints using both stratified sampling and our approach. For stratified sampling, we created a 316-by-316 grid and set the strata sizes (the number of datapoints in each cell) as balanced as possible across the cells created by the grid. In the zoomed-out overview plots, the visualization quality of the two competing methods seem nearly identical; however, when a zoomed-in plot is generated, one can observe that our proposed method delivers significantly richer information.

Previous Approaches— Architecturally, our system is similar to ScalaR [29], which interposes a data reduction layer between the visualization tool and a database backend; however, that project uses simple uniform random sampling. Researchers have attempted a number of approaches for improving visualization time, including binned aggregation [110, 115, 175], parallel rendering [42, 142], and incremental visualization [51, 52]. These methods are orthogonal to the one we propose here.

Our Goals— This chapter tackles the technical challenge of creating a sampling strategy that will yield *useful and high-quality* scatter and map plots at *arbitrary zooming resolutions* with as *few sampled tuples* as possible. Figure 5.1(d) shows the plot generated by our proposed method, which we call Visualization-Aware Sampling (VAS). Using the same number of tuples as random and stratified sampling, VAS yields a much higher-fidelity result. The use of VAS can be specified as part of the queries submitted by visualization tools to the database. Using VAS, the database returns an approximate query answer within a specified time bound using one of multiple pre-generated samples. VAS chooses

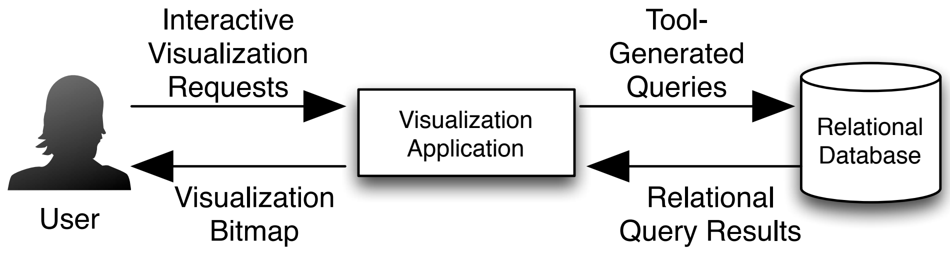


Figure 5.3: Standard model of user interaction with the combined visualization and database system.

an appropriate sample size by converting the specified time bound into the number of tuples that can likely be processed within that time bound. VAS is successful because it samples data points according to a visualization-oriented metric that correlates well with user success across a range of scatter and map plot tasks.

Contributions— We make the following contributions:

- We define the notion of VAS as an optimization problem (Section 5.3).
- We prove that the VAS problem is NP-hard and an offer efficient approximation algorithm. We establish a worst-case guarantee for our approximate solution (Section 5.4).
- In a user study, we show that our VAS is highly correlated with the user's success rate in various visualization tasks. We also evaluate the efficiency and effectiveness of our approximation algorithm over several datasets. We show that VAS can deliver a visualization that has equal quality with competing approaches, but using up to 400$\times$ fewer data points. Alternatively, if VAS can process an equal number of data points as competing methods, it can deliver a visualization with a significantly higher quality (Section 5.6).

Finally, we cover related work in Section 5.7 and conclude with a discussion of future work in Section 5.8.

5.2 System Overview

5.2.1 Software Architecture Model

Figure 5.3 shows the software architecture model that we focus on in this chapter. This is a standard architecture supported by the popular Tableau system [8]. It is also similar to ScalaR's "dynamic reduction" software architecture [29]. The user interacts with a visualization tool to describe a desired visualization — say, a scatterplot of Web server time vs latency. This tool has been configured to access a dedicated RDBMS, and the schema information from the RDBMS is visible in the user's interaction with the tool. For example, in Tableau, a user can partially specify a desired scatterplot by indicating a column name (say, server latency) from a list of options populated from the RDBMS metadata. The user must not only choose just which fields and ranges from the database are rendered, but also choose image-specific parameters such as visualization type, axis labels, color codings, and so on.

Once the user has fully specified a visualization, the tool requests the necessary data by generating the appropriate SQL query and submitting it to the remote RDBMS. The RDBMS then returns the (relational) query results back to the visualization tool. Finally, the tool uses the fetched records to render the final visualization bitmap, which is displayed to the user. During these last three steps, the user waits idly for the visualization tool and the RDBMS.

When large datasets are being visualized, extremely long waits can negatively affect the analyst's level of engagement and ability to interactively produce successive visualizations [28, 67, 114, 123, 157]. As reported in Figure 5.2, our own experiments show that the industry-standard Tableau tool can take more than four minutes to produce a scatterplot on just 50M tuples fetched from an in-memory database.

Note that our sampling approach is not limited to the software architecture in Figure 5.3, as reducing the number of visualized records almost always brings performance benefits. Thus, even if engineers decide to combine the visualization and data management layers in the future, sampling-based methods will still be useful.

5.2.2 Data Sampling

Approximate query processing via sampling is a popular technique [12, 16, 27, 38, 68, 81, 125, 131] for reducing the number of returned records, and random sampling or stratified sampling are two well-known methods for this. When using these methods, the visualization tool's query is run over a *sampled table(s)* (or simply, a *sample*) that is smaller than, and derived from, the original table(s). The sample(s) can be maintained by the same RDBMS. Since sampling approaches incur an additional overhead to produce the sample(s), these are typically performed in an *offline* manner [15, 38]: Once the sample is created and stored in the database, they can be interactively queried and visualized many times. (A sample can also be periodically updated when new data arrives [16].)

There is, of course, a tradeoff between output result quality and the size of the sample[1] (and thus, runtime). In the limit, a random sample of 100% of the original database will produce results with perfect fidelity, but will also not yield any reduction in runtime. Conversely, a sample of 0% of the database will yield a result with no fidelity, albeit very quickly. The exact size of the sample budget will be determined by deployment-specific details: the nature of the application, the patience of the user base, the amount of hardware resources available, and so on. As a result, the usefulness of a sampling method must be evaluated over a range of sample budgets, with a realistic measure of final output quality. Choosing a correct sampling budget is a known issue in the approximate query processing literature [15]. Of course, in any specific real-world deployment, we expect that the application will have a fixed maximum runtime or the size of a sample that the system must observe.

5.2.3 Visualization Quality

In this work, we focus on the production of *scatterplots* (including map plots, such as Figure 5.1) as one of the most popular visualization techniques. We leave other visualization types (such as bar charts, line charts, chloropleths, and so on) to future work.

Since the final value of a visualization is how much it helps the user, evaluating any sampling method means examining how users actually employ the visualizations they produce. Schneiderman, et al. proposed a taxonomy for information visualization types [158] and compiled some of common visualization-driven goals/tasks. Their list of goals included (i) regression, (ii) density estimation, and (iii) clustering. Our system aims to yield visualizations that help with each of these popular goals. We make no claim about other user goals and tasks for visualization, such as pattern finding and outlier detection, which we reserve for future work (although we have anecdotal evidence to suggest our system can address some of these tasks, too).

In this context, regression is the task of (visually) estimating the value of dependent variables given the value of independent variables. For example, if we want to know the temperature of the location specified by a pair of latitude and longitude coordinates, it belongs to the regression task. Density estimation is the task of understanding the distribution of the original data. For instance, one can use a map plot to understand the geometric area with the most cell phone subscribers. Clustering is a task that assigns data elements into distinct sets such that the data in the same group tend to be close to one another, while data in different groups are comparatively far apart.

[1]Here, the size of a sample means the number of the data points contained in the sample.

Schneiderman, et al.'s list also included goals/tasks that are either poor fits for scatter plots, or are simply outside the scope of what we aim to accomplish in this chapter: shape visualization (DNA or 3D structures), classification, hierarchy understanding, and community detection in networks. We explicitly do *not* attempt to produce visualizations that can help users with these tasks.

5.2.4 Our Approach

Our proposed method proceeds in two steps: (1) during *offline preprocessing*, we produce a sample that enable fast queries later, and (2) at *query time*, we choose a sample whose size is appropriate for the specific query.

Similar to any offline indexing technique, VAS also requires (1) the user to make choices about indexed columns and (2) an upfront computational work to speed up future queries. In other words, VAS can be considered as a specialized index designed for visualization workloads (e.g., Tableau). Note that, currently, even if users want to use offline indexing, there is no indexing technique that ensures fast and accurate visualizations, a problem solved by VAS.

Indexed columns can be chosen in three ways:

1. manually, by the DBA;
2. based on the most frequently visualized columns [15,73]; or
3. based on statistical properties of the data [135].

Among these approaches, the second one is the simplest, works reasonably well in practice, and can be made resilient against workload changes [126]. Furthermore, note that visualization workloads, especially those supported by BI tools and SQL engines, are similar to exploratory SQL analytics (i.e., grouping, filtering, aggregations). Real-world traces from Facebook and Conviva [15] reveal that 80-90% of exploratory queries use 5-10% of the column combinations. Moreover, VAS only requires frequently visualized column pairs, not groups or filters.

The core innovation of our work is that we generate a sample according to a *visualization-specific metric*. That is, we believe that when a sample is generated according to the metric we propose in Section 5.3 below, the user will be able to accomplish their goals from Section 5.2.3 above (i.e., regression, density estimation, clustering) using the resulting visualization, even with a small number of rendered data points. We do *not* claim that our method will work for other visualization types, or even for other visualization goals. Indeed, our method of generating a sample could in principle be harmful to some goals (such as community detection tasks that require all members of a latent set to be sampled). However, scatter plots, map plots, and the three visualization goals we focus on are quite widespread and useful. Furthermore, modifying a visualization tool to only use our sampling method if a user declares an interest in one of these goals would be a straightforward task.

5.3 Problem Formulation

The first step in our approach to obtain a good sample for scatter plot visualizations is defining a mathematical loss function that is closely correlated with the loss of visualization quality/utility from the user's perspective. Once we define a function that captures the visualization utility, our goal will be to solve an optimization problem; that is, finding a sample of a given size that has the minimum value for the loss function. In the rest of this section, we formally define our loss function and optimization problem. The algorithm for solving the derived optimization problem will be presented in Section 5.4. Also, in Section 5.6.2, we will report a comprehensive user study confirming that minimizing our loss function does indeed yield more useful visualizations for various user tasks.

We start our problem formulation by defining notations. We denote a dataset D of N tuples by $D = \{t_1, t_2, \ldots, t_N\}$. Each tuple t_i encodes the coordinate at which the associated *point* is displayed. For example, t_i is a pair of longitude and latitude in a map plot. A sample S is a subset of the dataset D and is denoted by $S = \{s_1, s_2, \ldots, s_K\}$. Naturally, s_i is one of t_j where $j = 1, \ldots, N$. The size of the sample S (which is denoted by K) is pre-determined based on the interactive latency requirement (see Section 5.2.2) and is given as an input to our problem.

In designing our loss function, the objective is to measure the visualization quality degradation originating from the sampling process. The traditional goal of sampling in database systems is to maximize the number of tuples that match a selection predicate, particularly those on categorical attributes [15]. In contrast, the selection predicates of a scatter/map plot are on a continuous range, for which traditional approaches (e.g., uniform or stratified sampling) may not lead to high quality visualizations.

Therefore, to develop a more visualization-focused sampling technique, we first imagine a 2D space on which a scatter plot is displayed, and let x denote any of the points on the space. To measure the visualization quality loss, we make the following observations:

1. The visualization quality loss occurs, as the sample S does not include all tuples of D.

2. The quality loss at x is reduced as the sample includes points at or near x — two plots drawn using the original dataset (D) and the sample (S) might not look identical if S does not include a point at x whereas D includes one at x, but they will look similar if the sample contains points near x.

3. When there are already many points at or near x, choosing more points in that neighborhood does not significantly enhance a visualization.

To express the above observations in a formal way, we consider the following measure for visualization quality degradation at the point x:

$$\text{point-loss}(x) = \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)}.$$

where $\kappa(x, s_i)$ is the function that captures the *proximity* between the two points, x and s_i. In this chapter, we use $\kappa(x, s_i) = \exp(-\|x - s_i\|^2/\epsilon^2)$ (see footnote[2] for ϵ) although other functions can also be used for κ if the function is a decreasing convex function of $\|x - s_i\|$ — the convexity is needed due to the third observation we described above. The equivalent quality metric can also be obtained by considering the problem as the regression problem that aims to approximate the original tuples in D using the sample S. See our technical report for an alternative derivation [136]. Note that the above loss value is reduced if there exists more sampled points near x where the proximity to x is captured by $\kappa(x, s_i)$. In other words, the visualization quality loss at x is minimized if S includes as many points as possible at and around x.

Note that the above loss function is defined for a single point x on the space on which a scatter/map plot is visualized, whereas the space on which a scatter plot is drawn has many of those points. As a result, our goal should be to obtain a sample S that minimizes the *combined* loss of all possible points on the space. Due to the reason, our objective is to find a sample S that minimizes the following expression:

$$\text{Loss}(S) = \int \text{point-loss}(x)\, dx = \int \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)}\, dx \tag{5.1}$$

Here, the integration is performed over the entire 2D space.

Now, we perform several mathematical tricks to obtain an effectively equivalent but a more tractable problem, because the exact computation of the above integration requires an computationally expensive method such as a Monte Carlo experiment with a large

[2]In our experiments, we set $\epsilon \approx \max(\|x_i - x_j\|)/100$ but there is a theory on how to choose the optimal value for ϵ as the only unknown parameter [39].

number of points. Using a second-order Taylor expansion, we can obtain a simpler form that enables an efficient algorithm in the next section:

$$\begin{aligned}
&\min \int \frac{1}{\sum_{s_i \in S} \kappa(x, s_i)} \, dx \\
&= \min \int 1 - (\sum \kappa(x, s_i) - 1) + (\sum \kappa(x, s_i) - 1)^2 \, dx \\
&= \min \int (\sum \kappa(x, s_i))^2 - 3 \sum \kappa(x, s_i) \, dx \\
&= \min \int \sum_{s_i, s_j \in S} \kappa(x, s_i) \kappa(x, s_j) \, dx
\end{aligned}$$

To obtain the last expression, we used the fact that the term $\int \sum \kappa(x, s_i) \, dx$ is constant since $\kappa(x, s_i)$ is a similarity function and we are integrating over every possible x, i.e., $\int \sum \kappa(x, s_i) \, dx$ has the same value regardless of the value of s_i. For the same reason, $\int \sum [\kappa(x, s_i)]^2 \, dx$ is also constant. By changing the order of integration and summation, we obtain the following optimization formulation, which we refer to as Visualization-Aware Sampling (VAS) problem in this chapter.

Definition 9 (VAS)**.** Given a fixed K, VAS is the problem of obtaining a sample S of size K as a solution to the following optimization problem:

$$\min_{S \subseteq D;\, |S| = K} \sum_{s_i, s_j \in S;\, i < j} \tilde{\kappa}(s_i, s_j)$$

$$\text{where } \tilde{\kappa}(s_i, s_j) = \int \kappa(x, s_i) \kappa(x, s_j) dx$$

In the definition above, we call the summation term $\sum \tilde{\kappa}(s_i, s_j)$ the *optimization objective*. With our choice of the proximity function, $\kappa(s_i, s_j) = \exp(-\|s_i - s_j\|^2 / \epsilon^2)$, we can obtain a concrete expression for $\tilde{\kappa}(s_i, s_j)$: $\exp(-\|s_i - s_j\|^2 / 2\epsilon^2)$, after eliminating constant terms that do not affect the minimization problem. In other words, $\tilde{\kappa}(s_i, s_j)$ is in the same form as the original proximity function. In general, $\tilde{\kappa}(s_i, s_j)$ is another proximity function between the two points s_i and s_j since the integration for $\tilde{\kappa}(s_i, s_j)$ tends to have a larger value when the two points are close. Thus, in practice, it is sufficient to use any proximity function directly in place of $\tilde{\kappa}(s_i, s_j)$.

In the next section, we show that the VAS problem defined above is NP-hard and we present an efficient approximation algorithm for solving this problem. Later in Section 5.6.2 we show that by finding a sample S that minimizes our loss function, we obtain a sample that, when visualized, best allows users to perform various visualization tasks.

5.4 Solving VAS

In this section, we focus on solving the optimization problem derived in the previous section to obtain an optimal sample S. In the following section, we also describe how to extend the sample obtained by solving VAS to provide a richer set of information.

5.4.1 Hardness of VAS

First, we analyze the hardness of VAS formally.

Theorem 5.1. VAS (Problem 9) is NP-hard.

Proof. We show the NP-hardness of Problem 9 by reducing *maximum edge subgraph* problem to VAS.

Lemma 6. (*Maximum Edge Subgraph*) Given a undirected weighted graph $G = (V, E)$, choose a subgraph $G' = (V', E')$ with $|V'| = K$ that maximizes

$$\sum_{(u,v) \subset E'} w(u, v)$$

This problem is called *maximum edge subgraph*, and is NP-hard [50].

To reduce the above problem to VAS, the following procedure is performed: map i-th vertex v_i to i-th instance x_i, and set the value of $\tilde{\kappa}(x_i, x_j)$ to $w_{max} - w(v_i, v_j)$, where $w_{max} = \max_{v_i, v_j \subset V'} w(v_i, v_j)$. The reduction process takes $O(|E| + |V|)$. Once the set of data points that minimize $\sum_{s_i, s_j \in X} \tilde{\kappa}(s_i, s_j)$ is obtained by solving VAS, we choose a set of corresponding vertices, and return them as an answer to the *maximum edge subgraph* problem. Since the *maximum edge subgraph* problem is NP-hard, and the reduction process takes a polynomial time, VAS is also NP-hard. □

Due to the NP-hardness of VAS, obtaining an exact solution to VAS is prohibitively slow, as we will empirically show in Section 5.6.4. Thus, in the rest of this section, we present an approximation algorithm for VAS (Section 5.4.2), followed by additional ideas for improvement (Section 5.4.2).

5.4.2 The Interchange Algorithm

In this section, we present our approximation algorithm, called Interchange. The Interchange algorithm starts from a randomly chosen set of size K and performs a replacement operation with a new data point if the operation decreases the optimization objective (i.e., the loss function). We call such a replacement, i.e., one that decreases the optimization objective, a *valid replacement*. In other words, Interchange tests for valid replacements as it sequentially reads through the data points from the dataset D.

One way to understand this algorithm theoretically is by imagining a Markov network in which each state represents a different subset of D where the size of the subset is K. The network then has a total of $\binom{D}{K}$ states. The transition between the states is defined as an exchange of one of the elements in the current subset S with another element in $D - S$. It is easy to see that the transition defined in this way is *irreducible*, i.e., any state can reach any other states following the transitions defined in such a way. Because Interchange is a process that continuously seeks a state with a lower optimization objective than the current one, Interchange is a hill climbing algorithm in the network.

Expand/Shrink procedure— Now we state how we can efficiently perform valid replacements. One approach to finding valid replacements is by substituting one of the elements in S with a new data point whenever one is read in, then computing the optimization objective of the set. For this computation, we need to call the proximity function $O(K^2)$ times as there are K elements in the set, and we need to compute a proximity function for every pair of elements in the set. This computation should be done for every element in S. Thus, to test for valid replacements, we need $O(K^3)$ computations for every new data point.

A more efficient approach is to consider only the part of the optimization objective for which the participating elements are *responsible*. We formally define the notion of responsibility as follows.

Definition 10. (*Responsibility*) The responsibility of an element s_i in set S is defined as:

$$\mathrm{rsp}_S(s_i) = \frac{1}{2} \sum_{s_j \in S, j \neq i} \tilde{\kappa}(s_i, s_j).$$

Using the responsibility, we can speed up the tests for valid replacements in the following way. Whenever considering a new data point t, take an existing element s_i in S, and compute the responsibility of t in the set $S - \{s_i\} + \{t\}$. This computation takes $O(K)$ times. It is easy to see that if the responsibility of t in $S - \{s_i\} + \{t\}$ is smaller than the responsibility of s_i in the original set S, the replacement operation of s_i with the new data point t is a *valid replacement*. In this way, we can compare the responsibilities without computing all pairwise proximity functions. Since this test should be performed for every element in S, it takes a total of $O(K^2)$ computations for every new data point.

Algorithm 4: Interchange algorithm.

```
input  : $D = \{t_1, t_2, \ldots, t_N\}$
output: A sample $S$ of size $K$

// set for pairs of (item, responsibility)
$R \leftarrow \varnothing$
foreach $t_i \in D$ do
    if $|R| < K$ then  $R \leftarrow$ Expand $(R, t_i)$
    else
        $R \leftarrow$ Expand $(R, t_i)$
        $R \leftarrow$ Shrink $(R)$
    end
end
$S \leftarrow$ pick the first item of every pair in $R$
return $S$

subroutine Expand $(R, t)$
    rsp $\leftarrow 0$      // responsibility
    foreach $(s_i, r_i) \in R$ do
        $l \leftarrow \tilde{\kappa}(t, s_i)$
        $r_i \leftarrow r_i + l$
        rsp $\leftarrow$ rsp $+ l$
    end
    insert $(t,$ rsp$)$ into $R$
    return $R$
end

subroutine Shrink $(R)$
    remove $(t, r)$ with largest $r$ from $R$
    foreach $(s_i, r_i) \in R$ do
        $r_i \leftarrow r_i - \tilde{\kappa}(t, s_i)$
    end
    return $R$
end
```

However, it is possible to make this operation even faster. Instead of testing for valid replacements by substituting the new data point t for one of the elements in S, we simply *expand* the set S by inserting t into the set, temporarily creating a set of size $K+1$. In this process, the responsibility of every element in S is updated accordingly. Next, we find the element with the largest responsibility in the expanded set and remove that element from the set, shrinking the set size back to K. Again, the responsibility of every element in S should be updated. Algorithm 4 shows the pseudo-code for this approach. The theorem below proves the correctness of the approach.

Theorem 5.2. For $s_i \in S$, if replacing s_i with a new element t reduces the optimization objective of S, applying `Expand` followed by `Shrink` in Algorithm 4 replaces s_i with t. Otherwise, S remains the same.

Proof. Let $\tilde{\kappa}(S)$ indicate $\sum_{s_i, s_j \in S, i<j} \tilde{\kappa}(s_i, s_j)$. Also, define $S_- = S - \{s_i\}$ and $S_+ = S + \{t\}$. We show that if the optimization objective before the replacement, namely $\tilde{\kappa}(S_- + \{s_i\})$, is larger than the optimization objective after the replacement, namely $\tilde{\kappa}(S_- + \{t\})$, then the responsibility of the existing element s_i in an expanded set, $\text{rsp}_{S_+}(s_i)$, is also larger than the responsibility of the new element t in the expanded set, $\text{rsp}_{S_+}(t)$. The proof is as follows:

$$\begin{aligned}
&\tilde{\kappa}(S_- + \{s_i\}) > \tilde{\kappa}(S_- + \{t\}) \\
\Longleftrightarrow\ &\sum_{s_j \in S_-} \tilde{\kappa}(s_i, s_j) > \sum_{s_j \in S_-} \tilde{\kappa}(t, s_j) \\
\Longleftrightarrow\ &\tilde{\kappa}(s_i, t) + \sum_{s_j \in S_-} \tilde{\kappa}(s_i, s_j) > \tilde{\kappa}(s_i, t) + \sum_{s_j \in S_-} \tilde{\kappa}(t, s_j) \\
\Longleftrightarrow\ &\text{rsp}_{S_+}(s_i) > \text{rsp}_{S_+}(t).
\end{aligned}$$

Since the responsibility of s_i is larger than that of t in the expanded set S_+, the `Shrink` routine will remove s_i. If no element exists whose responsibility is larger than that of t, then t is removed by this routine and S remains the same. $\square$

In both the `Expand` and `Shrink` routines, the responsibility of each element is updated using a single loop, so both routines take $O(K)$ computations whenever a new data point is considered. Thus, scanning the entire dataset and applying these two routines will take $O(NK)$ running time.

The Interchange algorithm, if it runs until no replacement decreases the optimization objective, has the following theoretical bound.

Theorem 5.3. Let's say that the sample obtained by Interchange is S_{int}, and the optimal sample is S_{opt}. The quality of S_{int}, or the optimization objective, has the following upper bound:

$$\frac{1}{K(K-1)} \sum_{s_i, s_j \in S_{int};\, i<j} \tilde{\kappa}(s_i, s_j)$$
$$\leq \frac{1}{4} + \frac{1}{K(K-1)} \sum_{s_i, s_j \in S_{opt};\, i<j} \tilde{\kappa}(s_i, s_j)$$

In the expression above, we compare the difference between the averaged optimization objectives.

Proof. Due to the submodularity of VAS, which we show in our technical report [136], we can apply the result of Nemhauser, et al. [129] and obtain the result above. □

Ideally, Interchange should be run until no more valid replacements are possible. However, in practice, we observed that even running the algorithm for half an hour produces a high quality sample. When more time is permitted, the algorithm will continuously improve the sample quality until convergence.

Speed-Up using the Locality of Proximity function— Proximity functions such as $\exp(-\|x-y\|^2/\epsilon^2)$ have a property called *locality*. The locality property of a proximity function indicates that its value becomes negligible when the distance between the two data points is not close—an idea also used in accelerating other algorithms [98]. For example, our proximity function value is 1.12×10^{-7} when the distance between the two points is 4ϵ; thus, even though we ignore pairs whose distance is larger than a certain threshold, it will not affect the final outcome much. Exploiting this property, we can make the `Expand` and `Shrink` operations much faster by only considering the data points that are close enough to new data points. For a proximity check, our implementation used R-tree.

5.5 Extending VAS: Embedding Density

VAS aims to minimize a visualization-driven quality loss, yielding scatter/map plots that are highly similar to those generated by visualizing the entire dataset. However, we need a different strategy if the user's intention is to estimate the density or find clusters from the scatter plot. This is because humans cannot visually distinguish multiple data points on a scatter plot if they are duplicates or extremely close to one another. This can make it hard to visually estimate the number or density of such data points. One way to address this is to account for the number of near-duplicate points in each region. For example, points drawn from a dense area can be plotted with a larger legend size or some jitter noise can be used to provide additional density in the plot. In other words, the font size of each point or the amount of jitter noise will be proportional to the original density of the region the point is drawn from. VAS can be easily extended to support such an approach, as follows:

1. Obtain a sample using our algorithm for VAS.
2. Attach a counter to every sampled point.
3. While scanning the dataset once more, increase a counter if its associated sampled point is the nearest neighbor of the data point that was just scanned.

With these extra counters, we can now visualize the density of areas (each of which is represented by its nearest sampled point), e.g., using different dot sizes or by adding jitter noise in proportion to each point's density. (See Section 5.6.2 for a scatter plot example.)

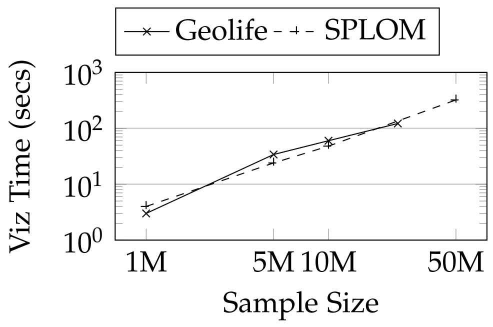
Note that the above process only adds an extra column to the database and, therefore, does not alter our basic Interchange algorithm for VAS. Also, this extension does not require any additional information from users.

Note that, for the above density embedding process, a special data structure such as a k-d tree [30] can be used to make the nearest neighbor tests more efficient. This is done by using the sample obtained in the first pass to build a k-d tree, then using the tree to identify the nearest data points in the sample during the second pass. Since k-d trees perform the nearest neighbor search in $O(\log K)$, the overall time complexity for the second pass is $O(N \log K)$.

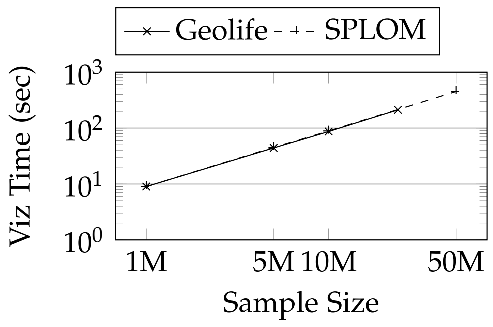
5.6 Experiments

We run four types of experiments to demonstrate that VAS and VAS with density embedding can produce high-quality plots in less time than competing methods.

1. We study the runtime of existing visualization systems that were introduced in Figure 5.2.
2. In a user study, we show that users were more successful when they used visualizations produced by VAS than with other methods. We also show that user success and our loss function were negatively correlated (that is, users were successful when our loss function is minimized).



(a) Tableau



(b) MathGL

Figure 5.4: Time to produce plots of various sizes using existing visualization systems.

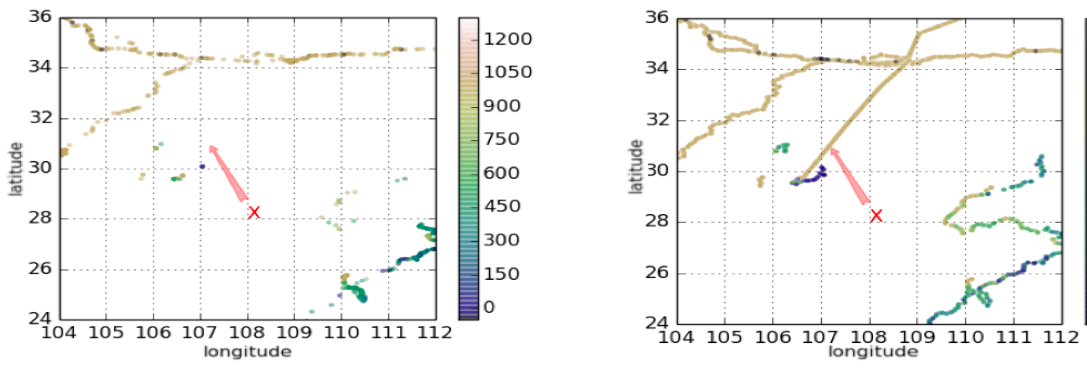
3. We show that VAS could obtain a sample of a fixed quality level (that is, loss function level) with fewer data points than competing methods. We demonstrate this over a range of different datasets and sample quality levels.
4. We empirically study the Interchange algorithm: we compare its quality and runtime to those of the exact method, examine the relationship between runtime and sample quality, and investigate the impact of our optimization on runtime.

All of our experiments were performed using two datasets: the Geolife dataset and the SPLOM dataset. The Geolife dataset was collected by Microsoft Research [189]. It contained latitude, longitude, elevation triples from GPS loggers, recorded mainly around Beijing. Our full database contained 24.4M tuples. We also used SPLOM, a synthetic dataset generated from several Gaussian distributions that had been used in previous visualization projects [88, 115]. We used parameters identical to previous work, and generated a dataset of five columns and 1B tuples. We performed our evaluations on an Amazon EC2 memory instance (r3.4xlarge) which contained 16 cores and 122G memory.

5.6.1 Existing Systems are Slow

Our discussions in this chapter are based on the idea that plotting a scatter plot using an original dataset takes an unacceptably long period of time. We tested two state-of-the-art systems: Tableau [9] and MathGL [120]. Tableau is one of the most popular commercial visualization software available on Windows, and MathGL is an open source scientific plotting library implemented in C++. We tested both the Geolife and SPLOM datasets. The results are shown in Figure 5.4.

In both systems, the visualization time includes (1) the time to load data from SSD storage (for MathGL) or from memory (for Tableau) and (2) the time to render the data into a plot. We can see that even when the datasets contained just 1M records, the



(a) Stratified Sampling

(b) VAS

Figure 5.5: Example figures used in the user study for the regression task. We asked the altitude of the location pointed by 'X'. The left was generated by stratified sampling and the right was generated by VAS.

visualization time was more than the 2-second interactive limit. Moreover, visualization time grew linearly with sample size.

5.6.2 User Success and Sample Quality

In this section we make two important experimental claims about user interaction with visualizations produced by our system. First, users are more successful at our specified goals when using VAS-produced outputs than when using outputs from uniform random sampling or stratified sampling. Second, user success and our loss function — that is, our measure of sample quality — are correlated. We validate these claims with a user study performed using Amazon's Mechanical Turk system.

User Success

We tested three user goals: regression, density estimation, and clustering.

Regression— To test user success in the regression task, we gave each user a sampled visualization from the Geolife data. We asked the users to estimate the altitude at a specified latitude and longitude. Naturally, the more sampled data points that are displayed near the location in question, the more accuracy users are likely to achieve. Figure 5.5 shows two examples of test visualizations given to users for the regression task (users were asked to estimate the altitude of the location marked by 'X'). We gave each user a list of four possible choices: the correct answer, two false answers, and "I'm not sure".

We tested VAS, random uniform sampling, and stratified sampling. We generated a test visualization for each sampling method at four distinct sample sizes ranging from

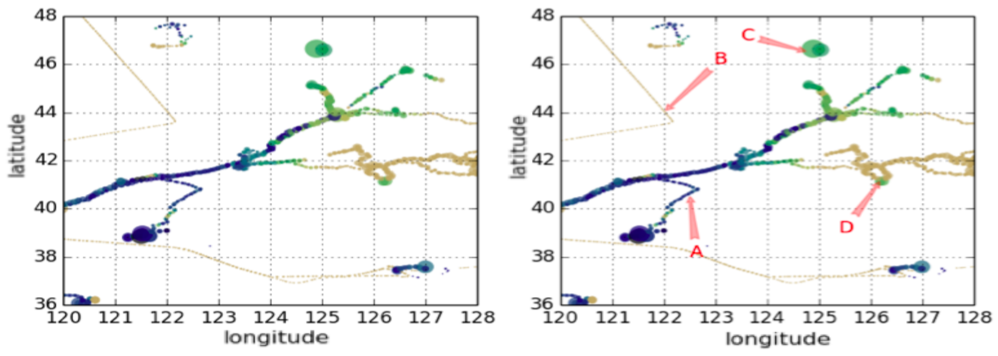


Figure 5.6: An example figure used in the user study for the density estimation task. This figure was generated using VAS with density embedding. The left-hand image is the original figure. The right-hand image contains four test markers, used to ask users to choose the densest area and the sparsest areas.

100 to 100K. For each test visualization, we zoomed into six randomly-chosen regions and picked a different test location for each region. Thus, we had 72 unique test questions (3 methods * 4 sample sizes * 6 locations). We gave each package of 72 questions to 40 different users and averaged the number of correct answers over each distinct question. To control for worker quality, we filtered out users who failed to correctly answer a few trivial "trapdoor" questions.

The uniform random sampling method chooses K data points purely at random, and as a result, tends to choose more data points from dense areas. We implemented the single-pass reservoir method for simple random sampling. Stratified sampling divides a domain into non-overlapping bins and performs uniform random sampling for each bin. Here, the number of the data points to draw for each bin is determined in the most balanced way. For example, suppose there are two bins and we want a sample of size 100. If there are enough data points to sample from those two bins, we sample 50 data points from each bin. Otherwise, if the second bin only has 10 available data points, then we sample 90 data points from the first bin, and 10 data points from the second bin. Stratified sampling is a straightforward method that avoids uniform random sampling's major shortcoming (that is, uniform random sampling draws most of its data points from the densest areas). In our experiment, stratified sampling divided the domain of Geolife into 100 exclusive bins and performed uniform random sampling for each bin using the reservoir method.

Table 5.1(a) summarizes user success in the regression task. The result shows that users achieved the highest accuracy in the regression task when they used VAS, significantly outperforming other sampling methods.

Sample size	Uniform	Stratified	VAS
100	0.213	0.225	0.428
1,000	0.260	0.285	0.637
10,000	0.215	0.360	0.895
100,000	0.593	0.644	0.989
Average	0.319	0.378	**0.734**

(a) Regression

Sample size	Uniform	Stratified	VAS	VAS w/ density
100	0.092	0.524	0.323	0.369
1,000	0.628	0.681	0.311	0.859
10,000	0.668	0.715	0.499	0.859
100,000	0.734	0.627	0.455	0.869
Average	0.531	0.637	0.395	**0.735**

(b) Density Estimation

Sample size	Uniform	Stratified	VAS	VAS w/ density
100	0.623	0.486	0.521	0.727
1,000	0.842	0.412	0.658	0.899
10,000	0.931	0.543	0.845	0.950
100,000	0.897	0.793	0.864	0.965
Average	0.821	0.561	0.722	**0.887**

(c) Clustering

Table 5.1: User Performance in Three Tasks

Density Estimation— For the density estimation task, we created samples whose sizes ranged 100-100K using four different sampling methods: uniform random sampling, stratified sampling, VAS, and VAS with density embedding. Using those samples, we chose 5 different zoomed-in areas. For each zoomed-in area, we asked users to identify the densest and the sparsest areas among 4 different marked locations. Figure 5.6 shows an example visualization shown to a test user. As a result, we generated 80 unique visualizations. We again posed the package of 80 questions to 40 unique users, and again filtered out users who failed to answer easy trapdoor questions.

The result of the density estimation task is shown in Table 5.1(b). Interestingly, the basic VAS method *without* density estimation yielded very poor results. However, when we augmented the sample with density embedding, users obtained even better success than with uniform random sampling. One of the reasons that 'VAS with density' was superior to uniform random sampling was because we not only asked the users to estimate the densest area, but also asked them to estimate the sparsest area of those figures. The figures generated by uniform random sampling typically had few points in sparse areas, making it difficult to identify the sparsest area.

Clustering— Lastly, we compared user performance in the clustering task. Since the Geolife dataset did not have ground-truth for clustering, we used synthetic datasets that we generated using Gaussian distributions instead. Using two-dimensional Gaussian distributions with different covariances, we generated 4 datasets, 2 of which were generated from 2 Gaussian distributions and the other 2 were generated from a single Gaussian distribution. (This dataset was similar to SPLOM, which unfortunately has a single Gaussian cluster, making it unsuitable for this experiment.)

Using the same 4 sampling methods that were used in the density estimation task, we created samples whose sizes ranged 100-100K, and tested if users could correctly identify the number of underlying clusters given the figures generated from those samples. In total, we created 64 questions (4 methods, 4 datasets, and 4 sample sizes). We again asked 40 Mechanical Turk users (or simply Turkers) and filtered out bad workers.

Table 5.1(c) summarizes the result of the clustering task. As in the density estimation task, 'VAS with density' allowed users to be more successful than they were with visualizations from uniform random sampling. Although VAS without density did not perform as well as uniform random sampling, it produced a roughly comparable score.

We think the reason VAS *without* density estimation showed comparable performance was that we used no more than 2 Gaussian distributions for data generation, and the Turkers could recognize the number of the underlying clusters from the outline of the sampled data points. For example, if the data were generated from two Gaussian distributions, the data points sampled by VAS would look like two partially overlapping circles. The Turkers would have shown poorer performance if there was a cluster surrounded by other clusters.

On the other hand, stratified sampling did poorly in this clustering task because it performed a separate random sampling for each bin, i.e., the data points within each bin tend to group together, and as a result, the Turkers found that there were more clusters than actually existed.

Correlation with Sample Quality

In this section, we test whether the VAS's optimization criterion of $\mathrm{Loss}(S)$ had a close relationship to our visualization users' success in reaching their end goals. If they were highly correlated, we have some empirical evidence that samples which minimize the VAS loss function will yield useful plots.

In particular, we examined this relationship for the case of regression. For each combination of sample size and sampling method, we produced a sample and corresponding visualization. We computed $\text{Loss}(S)$ using the expression in Equation 5.1. We then measured the correlation between the *loss* and average user performance on the regression task for that visualization.

To compute the loss (Equation 5.1), which includes integration, we used the Monte Carlo technique using 1,000 randomly generated points in the domain of the Geolife dataset. For this process, we determined that randomly generated points were within the domain if there existed any data point in the original dataset whose distance to the randomly generated data points was no larger than 0.1. Now, the integral expression was replaced with a summation as follows:

$$\text{Loss}(S) = \frac{1}{1000} \sum_{i=1}^{1000} \frac{1}{\sum_{s_i \in S} \kappa(x_i, s_i)}.$$

This *loss* computed above is the *mean* of one thousand values. One problem we encountered in computing the mean was that the point-loss often became so large that `double` precision could not hold those quantities. To address this, we used the *median* instead of the *mean* in this section because the median is less sensitive to outliers. Note that the median is still valid for a correlation analysis because we did not observe any case where a sample with a larger mean has a smaller median compared to another sample.

Next, to compare loss in a more consistent way, we computed the following quantity:

$$\text{log-loss-ratio}(S) = \log_{10} \left[\frac{\text{Loss}(S)}{\text{Loss}(D)} \right]$$

where D is the original dataset. $\text{Loss}(D)$ is the lowest *loss* that a sample can achieve; thus, samples with log-loss-ratios close to zero can be regarded as good samples based on this metric.

Next we examined the relationship between a sample's log-loss-ratio and the percentage of the questions that were correctly answered in the regression task using the sample's corresponding visualization. If the two metrics yield similar rankings of the sampled sets, then the VAS optimization criterion is a good guide for producing end-user visualizations. If the two metrics yield uncorrelated rankings, then our VAS criterion is faulty.

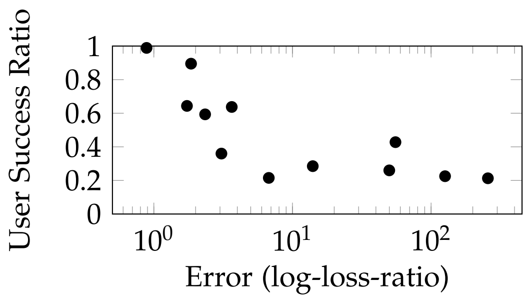


Figure 5.7: The relationship between the loss and user performance on the regression task. The samples with smaller losses resulted in better success ratios in general in the regression task.

Figure 5.7 depicts the results. The figure clearly shows the negative correlation between the loss and user success ratio in the regression task. Because the X-axis of the figure is the loss function that we aim to minimize to obtain a good sample, the negative correlation between the two metrics shows the validity of our problem formulation.

Also, when we computed Spearman's rank correlation coefficient[3], the correlation coefficient was -0.85, indicating a strong negative correlation between user success and the log-loss-ratio. (Its p-value was 5.2×10^{-4}.) Put another way, minimizing our loss function for a sample should do a good job of maximizing user success on the resulting visualizations. This result indicates that the problem formulation in Section 5.3 and the intuition behind it was largely valid.

5.6.3 VAS Uses a Smaller Sample

This section shows that VAS can produce a better sample than random uniform sampling or stratified sampling. That is, for a fixed amount of visualization production time, its quality (loss function value) is lower; or, that for a fixed quality level (loss function value), it needs less time to produce the visualization. (The visualization production time is linear with the number of data points.)

We used the Geolife dataset and produced samples of various sizes (and thus, different visualization production times). Figure 5.8(a) shows the results when we varied the visualization time: VAS always produced a sample with lower loss function values (i.e., higher quality) than other methods. The quality gap between the methods did not become smaller until after an entire minute of runtime. We show a similar result with the other dataset in our technical report [136]

[3]Spearman's rank correlation coefficient produces -1.0 for pairs of variables that are completely negatively correlated, and 1.0 for pairs of variables that are completely positively correlated.

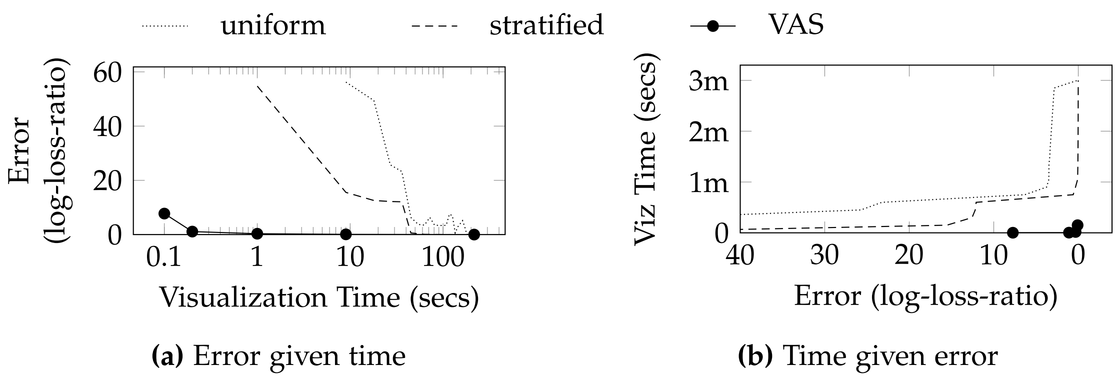


Figure 5.8: Relationship between visualization production time and error for the three sampling methods.

Figure 5.8(b) shows the same data using a different perspective. We fixed the loss function value (quality) and measured how long it takes to visualize the corresponding samples. Because the samples generated by our method had much smaller losses compared to other methods, all of the data points in the figure are in the bottom right corner. Competing methods required much more time than VAS to obtain the same quality (loss function value).

5.6.4 Algorithmic Details

We now examine three internal qualities of the VAS technique: approximate vs. exact solution, runtime analysis, and optimization contributions.

Exact vs. Approximate— The NP-hardness of VAS supports the need for an approximation algorithm. This section empirically examines the NP-hardness of VAS.

We think one of the best options for obtaining an exact solution to VAS is by converting the problem to an instance of integer programming and solving it using a standard library. Refer to our report [136] for converting VAS to an instance of Mixed Integer Programming (MIP). We used the GNU Linear Programming Kit [118] to solve the converted MIP problem.

Table 5.2 shows the time it took to obtain exact solutions to VAS with datasets of very small sizes. The sample size K was fixed to 10 in all of the experiments in the table. According to the result, the exact solutions to VAS showed better quality, but the procedure to obtain them took considerably longer. As shown, obtaining an exact solution when $N = 80$ took more than 40 minutes, whereas the time it took by other

N	Metric	MIP	Approx. VAS	Random
50	Runtime	1m 7s	0s	0s
	Opt. objective	0.160	0.179	3.72
	$Loss(S)$	1.5e+26	1.5e+26	2.5e+29
60	Runtime	1m 33s	0s	0s
	Opt. objective	0.036	0.076	3.31
	$Loss(S)$	3.8e+11	1.6e+16	2.5e+29
70	Runtime	14m 26s	0s	0s
	Opt. objective	0.047	0.048	3.02
	$Loss(S)$	1.8e+13	1.8e+13	9.45e+33
80	Runtime	48m 55s	0s	0s
	Opt. objective	0.043	0.048	2.25
	$Loss(S)$	8.5e+13	1.8e+13	9.4e+35

Table 5.2: Loss and runtime comparison

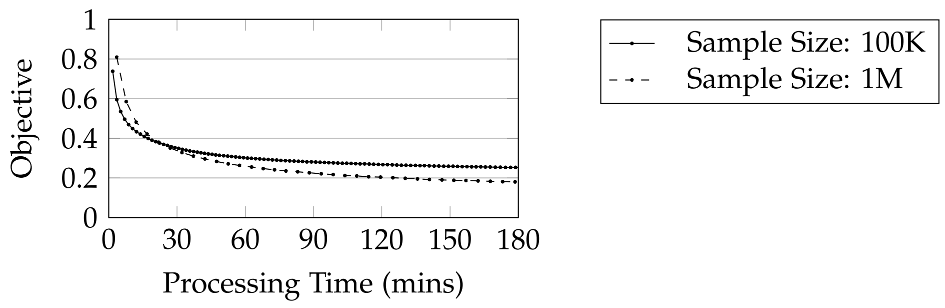
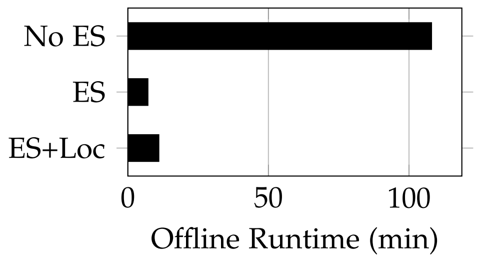


Figure 5.9: Processing Time vs. Quality. The lower the objective, the higher the quality is. The Interchange algorithm for VAS produces a high-quality visualization in a relatively short period of time. The quality is improved incrementally as more processing time is allowed.

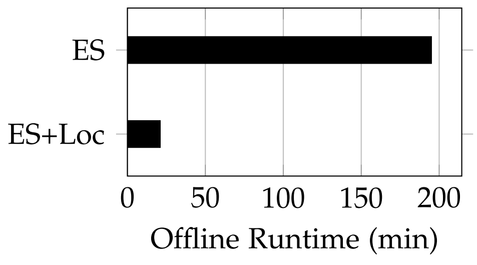
sampling methods was negligible. Clearly, the exact solution is not feasible except for extremely small data sizes.

Runtime Analysis— VAS gradually improves its sample quality as more data is read and processed. As observed in many statistical optimization routines, VAS offers good-quality plots long before reaching its optimal state. To investigate this phenomenon, we measured the relationship between "processing time" and "visualization quality." The result is shown in Figure 5.9. Note that the Y-axis of the figure is the objective[4] of our minimization problem; thus, the lower the objective, the higher the associated visualization's quality. In this experiment, we used the Geolife dataset. Figure 5.9 demonstrates that our Interchange algorithm improved the visualization quality quickly at its initial

[4]We scaled the objectives appropriately for a clearer comparison.



(a) Small Sample Size (100)



(b) Large Sample Size (5K)

Figure 5.10: Runtime comparison of different levels of optimizations. For this experiment, we used the Geolife dataset. ES+Loc indicates that both Expand/Shrink (ES) operation and the locality of a proximity function were used.

stages, and the improvement rate slowed down gradually. Notably, VAS produced low-error plots within only tens of minutes of processing time. The storage overhead of our algorithm is only $O(K)$, where K is the sample size.

Optimization Contribution— To quantify the impact of our optimization efforts on the runtime reduction, we measured the runtime of three different settings:

1. No Expand/Shrink (No ES): This is the most basic configuration that does not use the Expand/Shrink approach, but instead compares the responsibility when a new point is switched with another one in the sample.
2. Expand/Shrink (ES): This version uses the Expand/Shrink operation, reducing the time complexity by $O(K)$, where K is the sample size.
3. Expand/Shrink+Locality (ES+Loc): This version uses an additional R-tree to speed up the Expand/Shrink operations. This version is possible due to the locality of our loss function.

Figure 5.10 shows the results. When the sample size was relatively small (100), the second approach (Expand/Shrink), which does not exploit the locality, showed the shortest runtime due to no extra overhead coming from maintaining an extra R-tree data structure. However, when the sample size was relatively large (5K), the last approach (ES+Loc) that exploits the locality of the loss function showed the fastest runtime. When the user is interested in large samples (more than 10K at least), the last approach that uses R-tree to exploit locality will be the most preferable choice. The runtime sensitivity to sample size suggests that in the future, it may be useful to employ an optimizer that chooses the most appropriate algorithm setting, given a requested sample size.

5.7 Related Work

Support for interactive visualization of large datasets is a fast-growing area of research interest [28,29,42,43,51,52,67,110,115,142,175], along with other approximate techniques for interactive processing of non-conventional queries [137]. Most of the work to date has originated from the user interaction community, but researchers in database management have begun to study the problem. Known approaches fall into a few different categories.

The most directly related work is that of Battle, et al. [29]. They proposed ScalaR, a system for *dynamic reduction* of query results that are too large to be effectively rendered on-screen. The system examines queries sent from the visualization system to the RDBMS and if necessary, inserts aggregation, sampling, and filtering query operators. ScalaR uses simple random sampling, and so could likely be improved by adopting our sampling technique. For bar graphs, Kim et al. [96] proposed an order-preserving sampling method, which examines fewer tuples than simple random sampling.

Binned aggregation approaches [110, 115, 175] reduce data by dividing a data domain into tiles or bins, which correspond to materialized views. At visualization time, these bins can be selected and aggregated to produce the desired visualization. Unfortunately, the exact bins are chosen ahead of time, and certain operations — such as zooming — entail either choosing a very small bin size (and thus worse performance) or living with low-resolution results. Because binned aggregation needs to pre-aggregate all the quantities in advance, the approach is less flexible when the data is changing, such as measured temperatures over time; our method does not have such a problem.

Wickham [175] proposed to improve visualization times with a mixture of binning and summarizing (very similar to binned aggregation) followed by a statistical smoothing step. The smoothing step allows the system to avoid problems of high variability, which arise when the bins are small or when they contain eccentric values. However, the resulting smoothed data may make the results unsuitable for certain applications, such as an outlier finding. This smoothing step itself is orthogonal to our work, i.e., when there appears to be high variability in the sample created by our proposed method, the same smoothing technique can be applied to present more interpretable results. The smoothing process also benefits from our method because VAS creates a sample much smaller than the original database, thus, makes smoothing faster. The *abstract rendering pipeline* [43] also maps bins to regions of data, but the primary goal of this system is to modify the visualization, not performance.

Parallel rendering exploits parallelism in hardware to speed up visual drawing of the visualization [42,142]. It is helpful but largely orthogonal to our contributions. SeeDB is a system that discovers the most interesting bar graphs [170] from datasets.

Incremental visualization proposes a streaming data processing model, which quickly yields an initial low-resolution version of the user's desired bitmap [51,52]. The system continues to process data after showing the initial image and progressively refines the visualization. When viewed in terms of our framework in Section 5.2, this method amounts to increasing the sample budget over time and using the new samples to improve the user's current visualization. Thus, incremental visualization and sample-driven methods should benefit from each other.

5.8 Summary

We have described the VAS, a data reduction method for visualizations. VAS is able to choose a subset of the original database that is very small (and thus, fast) while still yielding a high-quality scatter or map plot. Our user study showed that for three common user goals — regression, density estimation, and clustering — VAS outputs are substantially more useful than other sampling methods' outputs with the same number of tuples.

The result of this chapter shows another possibility of speeding up AQP by building task-aware synopses. We believe the data analytics systems for visualization tools are still in its infancy and entails a range of interesting challenges. In particular, we plan to investigate techniques for rapidly generating visualizations for other user goals (including outlier detection, trend identification) and other data types (such as large networks). Task-aware synopses will be the key components in supporting those visualization tasks as well.

We have presented our two techniques—neighbor-sensitive hashing and visualization-aware sampling—which builds task-aware synopses for higher-quality data analytics. We end the second part of this dissertation.

Chapter 6

Conclusions and Future Work

Real-time data analytics has been the goal of many years of research in various communities. Since approximate query processing was proposed in the database community more than a decade ago, it has gained a renewed attention recently due to the rapid growth of data volume. If the current trend—faster growth in data volume compared to the growth in computational power—continues, approximate query processing will remain as an essential technique for real-time data analytics.

In this dissertation, we have shown that two approaches—exploiting past computations and building task-aware synopses—are effective for further speeding up approximate query processing. The proposed techniques in this dissertation cover important data analytics tasks: analytic SQL queries, data mining, and visualizations. Those analytics have been popular tools for data analysts and are projected to remain as important tools for analyzing data [147].

6.1 Lessons Learned

Approximate data analytics is like humans— Our database learning originates from the idea that, if data analytics systems were humans, how would they behave as answering queries? One similarity between AQP and humans is they show great ability in producing instant answers, albeit inexact. If we could employ thousands of humans for data analytics tasks instead of those computing devices, those human workers gradually acquire knowledge; thus, they can more quickly answer the future queries if those future queries are somewhat related to the queries they answered in the past.

Our prototype system, Verdict, formalizes this simple intuition based on probabilistic modeling of the answers to the SQL queries that include aggregate functions. As a result, our approach provides very natural ability as a learning agent. For instance, when past queries does not involve any common values with new queries for their aggregations and when those values are very weakly correlated, Verdict can barely improve the quality of the answers to new queries. This is analogous to asking a domain expert of a question unrelated to her expertises. Then, the domain expert would not be able to quickly answer the question.

We believe a similar idea can be applied to other types of data analytics tasks. Imagine one is interested in visualizing a massive graph. As a data visualization system processes many queries on many subparts of the graph, the system could gradually acquire more knowledge on various parts of the graph; thus, it would be able to combine its past visualizations somehow to generate a new visualization.

Target outliers to bound the worst-case latency— The primary reason of uniform random sampling's popularity would be its simplicity. However, uniform random sampling seriously priorities the items that belong to dense subpopulations. If our goal is, however, to lower the worst case query latency (e.g., to bound all query processing below 3 seconds), we must have enough sampled items from rare subpopulations. Our visualization-aware sampling is one way to achieve this goal by sampling data items in the most balanced way.

Approaches similar to visualization-aware sampling can be developed for other types of data analytics tasks to lower the worst-case performance. For instance, a similar approach could be used in place of the current stratified sampling for constructing samples in relational AQP databases. The current stratified sampling is effective only when the predefined strata coincides with the `group-by` attributes; however, an approach similar to visualization-aware sampling could benefit even when the `group-by` attributes of the future queries are not known in advance.

6.2 Future Work

As working on this dissertation, we have realized several interesting opportunities for applying and generalizing AQP. In this section, we present three interesting problems for future work.

6.2.1 Exploiting Past Computations for General SQL Processing

Data analysis with declarative languages is an important feature for both conventional RDMBS and modern distributed data analytics systems [2–4]. For those systems, accurate cardinality estimations are crucial for finding an optimal query plan. A study shows the errors in cardinality estimations caused more than 10× slowdowns for 8% of benchmark queries and more than 100× slowdowns for 5% of benchmark queries, for both open source and commercial database systems [103]. These slowdowns are more significant for large-scale data analysis; even a small relative efficiency difference can lead to a big absolute time difference.

In reality, accurate cardinality estimations are hard when (1) correlations exist among attributes or (2) a query selectivity is low. These conditions make commonly used static cardinality estimation techniques (such as per-attribute histograms [77] or sampling based methods [104, 111]) inaccurate. In general, high-accuracy estimations demand more histogram bins or more sampled tuples; however, larger numbers of bins and sampled tuples inherently cause larger memory footprints and increased estimation times. Given fixed budgets of space and time, static cardinality estimation techniques are essentially suboptimal unless query workloads are completely uniform over the entire data, or future query workloads are fully known in advance (so, those techniques are optimized for those workloads).

We can instead gradually improve the quality of selectivity estimations in a workload-sensitive manner by exploiting past computations, i.e., the selectivities of the past queries. The true selectivities are naturally obtained as byproducts of query processing. If database systems support AQP, approximate selectivities of the past queries will be obtained. Still, we can gradually refine our knowledge on the data distribution based on those selectivity information, which in turn can be used for estimating the selectivities of future queries.

6.2.2 AQP on Any Databases

Many AQP approaches have been proposed in the literature [11, 38, 68, 89, 184, 185], and we see several open-source implementations available [15, 185]. However, AQP database systems still not the first priority for big data analytics due to their limitation in supporting general SQL queries. Common choices for large-scale data analytics are the database systems based on distributed storage systems, such as Apache Hive, Apache Spark (and Spark SQL), and so on.

In contrast to existing approaches for AQP, i.e., dedicated AQP engines, we can build AQP functionality on top of existing large-scale database systems. For instance, for estimating an average and its confidence interval, we need to compute the average and the variance of the values using a sample table. Then, we can apply the central limit theorem (Chapter 2) to obtain the confidence interval. Note that both average and the variance of the values can be computed by obtaining the mean of the values and the mean of squared values. All we need is to obtain a sample of the original data and manage the information (e.g., location of the sample, sample size, etc.) in a separate metadata table. Then, when the user issues a query that can be approximately computed, an AQP module that exists on top of existing large-scale data analytics systems should redirect an rewritten query to the sample table (instead of the original table), and compose an answer to the query appropriately.

We believe this approach will be available on top of most of existing database systems, even including conventional databases, such as Oracle, IBM DB2, MySQL, etc. Then, the customers of those existing (and more stable due to longer history) database systems can enjoy AQP capability without modifying those existing systems. All they need is to issue an query to our module (instead of issuing the query directly to the database); then, our module will communicate with the underlying database and compute an approximate answer.

6.2.3 Diverse Approximate Analytics

Another problem in making AQP databases more general is the limitation in supported queries. Currently, most of existing systems support only the queries that include basic aggregate functions (e.g., `sum`, `avg`, `count`) without general joins and nested subqueries. When the `select` clause in SQL queries include complex expressions (e.g., `avg(discount) / avg(price * (1-discount))`) or when the queries include joins and nested subqueries, it is hard to derive confidence intervals of the approximate answers efficiently.

The goal of this project is to develop an efficient approach for computing the confidence intervals of such queries that may include complex aggregate expressions, joins, or nested subqueries. This development should also consider the previous project—AQP on any databases.

Bibliography

[1] https://db.apache.org/derby/docs/10.6/tuning/ctuntransform36368.html. Accessed: 2017-05-05.

[2] Apache hadoop. http://hadoop.apache.org/. Accessed: 2017-05-05.

[3] Apache hadoop. https://hive.apache.org/. Accessed: 2017-05-05.

[4] Apache impala (incubating). https://impala.incubator.apache.org/. Accessed: 2017-05-05.

[5] Apache spark. http://spark.apache.org/. Accessed: 2017-05-05.

[6] Presto: Distributed SQL query engine for big data. https://prestodb.io/docs/current/release/release-0.61.html. Accessed: 2017-05-05.

[7] SnappyData. http://www.snappydata.io/. Accessed: 2017-05-05.

[8] Tableau for the enterprise: An overview for it. http://www.tableausoftware.com/sites/default/files/whitepapers/whitepaper_tableau-for-the-enterprise_0.pdf. Accessed: 2017-05-05.

[9] Tableau software. http://www.tableausoftware.com/. Accessed: 2017-05-05.

[10] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, 1999.

[11] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[12] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The Aqua approximate query answering system. In *SIGMOD*, 1999.

[13] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, 1999.

[14] Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar, Michael Jordan, Samuel Madden, Barzan Mozafari, and Ion Stoica. Knowing when you're wrong: Building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[15] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[16] Sameer Agarwal, Aurojit Panda, Barzan Mozafari, Anand P. Iyer, Samuel Madden, and Ion Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 2012.

[17] Michael R Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[18] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, 2006.

[19] Dolan Antenucci, Michael R Anderson, and Michael Cafarella. A declarative query processing system for nowcasting. *PVLDB*, 2016.

[20] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.

[21] Michael Armbrust et al. Spark sql: Relational data processing in spark. In *SIGMOD*, 2015.

[22] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *SODA*, 2007.

[23] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The ts-tree: efficient time series search and retrieval. In *EDBT*, 2008.

[24] Vassilis Athitsos, Marios Hadjieleftheriou, George Kollios, and Stan Sclaroff. Query-sensitive embeddings. *TODS*, 2007.

[25] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, 2008.

[26] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *VLDB*, 2003.

[27] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.

[28] Mike Barnett, Badrish Chandramouli, Robert DeLine, Steven M. Drucker, Danyel Fisher, Jonathan Goldstein, Patrick Morrison, and John C. Platt. Stat!: an interactive analytics environment for big data. In *SIGMOD*, 2013.

[29] Leilani Battle, Michael Stonebraker, and Remco Chang. Dynamic reduction of query result sets for interactive visualizaton. In *BigData Conference*, pages 1–8, 2013.

[30] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975.

[31] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 1996.

[32] Christopher M Bishop. Pattern recognition. *Machine Learning*, 2006.

[33] Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *CVPR*, 2008.

[34] Erik Brynjolfsson, Lorin M Hitt, and Heekyung Hellen Kim. Strength in numbers: How does data-driven decisionmaking affect firm performance? 2011.

[35] Jaime G Carbonell, Ryszard S Michalski, and Tom M Mitchell. An overview of machine learning. In *Machine learning*. 1983.

[36] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[37] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

[38] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.

[39] Pavel Cizek, Wolfgang Karl Härdle, and Rafał Weron. *Statistical tools for finance and insurance*. Springer, 2005.

[40] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *NSDI*, 2010.

[41] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[42] Joseph A. Cottam and Andrew Lumsdaine. Automatic application of the data-state model in data-flow contexts. In *IV*, pages 5–10, 2010.

[43] Joseph A. Cottam, Andrew Lumsdaine, and Peter Wang. Overplotting: Unified solutions under abstract rendering. In *BigData Conference*, 2013.

[44] Bin Cui, Beng Chin Coi, Jianwen Su, and K-L Tan. Indexing high-dimensional data for efficient in-memory similarity search. *TKDE*, 2005.

[45] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, 2004.

[46] Amol Deshpande and Samuel Madden. Mauvedb: supporting model-based user views in database systems. In *SIGMOD*, 2006.

[47] Alin Dobra, Chris Jermaine, Florin Rusu, and Fei Xu. Turbo-charging estimate convergence in dbo. *PVLDB*, 2009.

[48] Amr El-Helw, Ihab F Ilyas, and Calisto Zuzarte. Statadvisor: Recommending statistical views. *VLDB*, 2009.

[49] Wenfei Fan, Floris Geerts, Yang Cao, Ting Deng, and Ping Lu. Querying big data by accessing small data. In *PODS*, 2015.

[50] Uriel Feige, David Peleg, and Guy Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29, 2001.

[51] Danyel Fisher, Steven M. Drucker, and Arnd Christian König. Exploratory visualization involving incremental, approximate database queries and uncertainty. *IEEE Computer Graphics and Applications*, 2012.

[52] Danyel Fisher, Igor O. Popov, Steven M. Drucker, and m. c. schraefel. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *CHI*, 2012.

[53] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004.

[54] D. Freedman, R. Pisani, and R. Purves. *Statistics*. 2007.

[55] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, 2012.

[56] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, 2000.

[57] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. Dsh: data sensitive hashing for high-dimensional k-nnsearch. In *SIGMOD*, 2014.

[58] Minos Garofalakis and Philip Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001. Tutorial.

[59] Wolfgang Gatterbauer and Dan Suciu. Approximate lifted inference with probabilistic databases. *PVLDB*, 2015.

[60] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *SIGMOD Record*, 2001.

[61] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[62] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *SIGARCH*, 2015.

[63] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[64] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[65] Alon Y Halevy. Answering queries using views: A survey. *VLDBJ*, 2001.

[66] Junfeng He, Regunathan Radhakrishnan, Shih-Fu Chang, and Claus Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, 2011.

[67] Jeffrey Heer and Sean Kandel. Interactive analysis of big data. *XRDS*, 2012.

[68] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD*, 1997.

[69] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *CVPR*, 2012.

[70] Katja Hose, Daniel Klan, and Kai-Uwe Sattler. Distributed data summaries for approximate query processing in pdms. In *IDEAS*, 2006.

[71] Ying Hu, Seema Sundara, and Jagannathan Srinivasan. Estimating Aggregates in Time-Constrained Approximate Queries in Oracle. In *EDBT*, 2009.

[72] Hai Huang, Chengfei Liu, and Xiaofang Zhou. Approximating query answering on rdf databases. *WWW*, 2012.

[73] Stratos Idreos, Martin L Kersten, and Stefan Manegold. Database cracking. In *CIDR*, 2007.

[74] Stratos Idreos, Martin L Kersten, and Stefan Manegold. Self-organizing tuple reconstruction in column-stores. In *SIGMOD*, 2009.

[75] Go Irie, Zhenguo Li, Xiao-Ming Wu, and Shih-Fu Chang. Locally linear hashing for extracting non-linear manifolds. In *CVPR*, 2014.

[76] Hosagrahar V Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *TODS*, 2005.

[77] HV Jagadish, Hui Jin, Beng Chin Ooi, and Kian-Lee Tan. Global optimization of histograms. *SIGMOD Record*, 2001.

[78] J. S. Roger Jang. General formula: Matrix inversion lemma. `http://www.cs.nthu.edu.tw/~jang/book/addenda/matinv/matinv/`.

[79] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *TPAM*, 2011.

[80] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *ICASSP*, 2011.

[81] Chris Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the dbo engine. *TODS*, 2008.

[82] Yuntao Jia. Running tpc-h queries on hive. https://issues.apache.org/jira/browse/HIVE-600.

[83] Zhongming Jin, Yao Hu, Yue Lin, Debing Zhang, Shiding Lin, Deng Cai, and Xuelong Li. Complementary projection hashing. In *ICCV*, 2013.

[84] Alexis Joly and Olivier Buisson. Random maximum margin hashing. In *CVPR*, 2011.

[85] Shantanu Joshi and Christopher Jermaine. Materialized sample views for database approximation. *TKDE*, 2008.

[86] Shantanu Joshi and Christopher Jermaine. Sampling-Based Estimators for Subset-Based Queries. *VLDB J.*, 2009.

[87] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan PC Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, et al. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 2008.

[88] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *AVI*, 2012.

[89] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, 2016.

[90] Shrikant Kashyap and Panagiotis Karras. Scalable knn search on vertically stored time series. In *SIGKDD*, 2011.

[91] Martin Kaufmann and Donald Kossmann. Storing and processing temporal data in a main memory column store. *PVLDB*, 2013.

[92] Raghav Kaushik, Christopher Ré, and Dan Suciu. General database statistics using entropy maximization. In *DBPL*, 2009.

[93] Dor Kedem, Stephen Tyree, Fei Sha, Gert R Lanckriet, and Kilian Q Weinberger. Non-linear metric learning. In *NIPS*, 2012.

[94] Alfons Kemper and Thomas Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, 2011.

[95] Daniel Keysers, Christian Gollan, and Hermann Ney. Local context in non-linear deformation models for handwritten character recognition. In *ICPR*, 2004.

[96] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 2015.

[97] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*. 2011.

[98] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 2008.

[99] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[100] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing. *TPAM*, 2012.

[101] Nikolay Laptev, Kai Zeng, and Carlo Zaniolo. Early Accurate Results for Advanced Analytics on MapReduce. *PVLDB*, 2012.

[102] Neil Lawrence, Matthias Seeger, Ralf Herbrich, et al. Fast sparse gaussian process methods: The informative vector machine. *NIPS*, 2003.

[103] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *PVLDB*, 2015.

[104] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.

[105] Christian Lemke, Kai-Uwe Sattler, Franz Faerber, and Alexander Zeier. Speeding up queries in column stores. In *DaWaK*, 2010.

[106] Michael S Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *TOMCCAP*, 2006.

[107] M. Lichman. UCI machine learning repository, 2013.

[108] King-Ip Lin, Hosagrahar V Jagadish, and Christos Faloutsos. The tv-tree: An index structure for high-dimensional data. *The VLDB Journal*, 1994.

[109] Yue Lin, Rong Jin, Deng Cai, Shuicheng Yan, and Xuelong Li. Compressed hashing. In *CVPR*, 2013.

[110] Lauro Didier Lins, James T. Klosowski, and Carlos Eduardo Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *TVCG*, 2013.

[111] Richard J Lipton, Jeffrey F Naughton, and Donovan A Schneider. *Practical selectivity estimation through adaptive sampling*. 1990.

[112] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[113] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *ICML*, 2011.

[114] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. 2002.

[115] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, 2013.

[116] Miodrag Lovric. *International Encyclopedia of Statistical Science.* Springer, 2011.

[117] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, 2007.

[118] A Makhorin. Glpk (gnu linear programming kit), version 4.54. *http://www.gnu.org/software/glpk*.

[119] Vikash Mansinghka et al. Bayesdb: A probabilistic programming system for querying the probable implications of data. *arXiv*, 2015.

[120] Mathgl. http://mathgl.sourceforge.net/doc_en/Main.html.

[121] Alexandra Meliou, Carlos Guestrin, and Joseph M Hellerstein. Approximating sensor network queries using in-network summaries. In *IPSN*, 2009.

[122] Charles A Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *JMLR*, 2006.

[123] Robert B Miller. Response time in man-computer conversational transactions. In *fall joint computer conference*, 1968.

[124] Renqiang Min, David Stanley, Zineng Yuan, Anthony Bonner, Zhaolei Zhang, et al. A deep non-linear feature mapping for large-margin knn classification. In *ICDM*, 2009.

[125] Barzan Mozafari. Verdict: A system for stochastic query planning. In *CIDR, Biennial Conference on Innovative Data Systems*, 2015.

[126] Barzan Mozafari, Eugene Zhen Ye Goh, and Dong Young Yoon. CliffGuard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

[127] Barzan Mozafari and Ning Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 2015.

[128] Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, and Kishor Bachhav. Snappydata: A unified cluster for streaming, transactions, and interactive analytics. In *CIDR*, 2017.

[129] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14, 1978.

[130] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, 2012.

[131] Christopher Olston, Edward Bortnikov, Khaled Elmeleegy, Flavio Junqueira, and Benjamin Reed. Interactive analysis of web-scale data. In *CIDR*, 2009.

[132] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.

[133] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, et al. The case for ramclouds: scalable high-performance storage entirely in dram. *SIGOPS*, 2010.

[134] Niketan Pansare, Vinayak R. Borkar, Chris Jermaine, and Tyson Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4, 2011.

[135] Aditya Parameswaran, Neoklis Polyzotis, and Hector Garcia-Molina. Seedb: Visualizing database queries efficiently. *PVLDB*, 2013.

[136] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. Technical report for visualization-aware sampling. `http://arxiv.org/abs/1510.03921`.

[137] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. Neighbor-sensitive hashing. *PVLDB*, 2015.

[138] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, 2016.

[139] Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, and Barzan Mozafari. Database learning: Toward a database that becomes smarter every time. In *SIGMOD*, 2017.

[140] Kasun S Perera, Martin Hahmann, Wolfgang Lehner, Torben Bach Pedersen, and Christian Thomsen. Efficient approximate olap querying over time series. In *IDEAS*, 2016.

[141] Eleni Petraki, Stratos Idreos, and Stefan Manegold. Holistic indexing in main-memory column-stores. In *SIGMOD*, 2015.

[142] Harald Piringer, Christian Tominski, Philipp Muigg, and Wolfgang Berger. A multi-threading architecture to support interactive visual exploration. *IEEE Trans. Vis. Comput. Graph.*, 2009.

[143] Hasso Plattner. A common database approach for oltp and olap using an in-memory column database. In *SIGMOD*, 2009.

[144] Abhijit Pol and Christopher Jermaine. Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, 2005.

[145] Navneet Potti and Jignesh M Patel. Daq: a new paradigm for approximate query processing. *PVLDB*, 2015.

[146] Jags Ramnarayan, Barzan Mozafari, Sudhir Menon, Sumedh Wale, Neeraj Kumar, Hemant Bhanawat, Soubhik Chakraborty, Yogesh Mahajan, Rishitesh Mishra, and Kishor Bachhav. Snappydata: A hybrid transactional analytical store built on spark. In *SIGMOD*, 2016.

[147] Philip Russom et al. Big data analytics. *TDWI best practices report, fourth quarter*, 2011.

[148] Florin Rusu, Chengjie Qin, and Martin Torres. Scalable analytics model calibration with online aggregation. *IEEE Data Eng. Bull.*, 2015.

[149] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 2009.

[150] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959.

[151] Harsimrat Sandhawalia and Hervé Jégou. Searching with expectations. In *ICASSP*, 2010.

[152] Sunita Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, 2000.

[153] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 2012.

[154] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*. 2007.

[155] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, 2003.

[156] Nikita Shamgunov. The memsql in-memory database system. In *IMDM@VLDB*, 2014.

[157] Ben Shneiderman. Response time and display rate in human performance with computers. *CSUR*, 1984.

[158] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages*, 1996.

[159] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *MSST*, 2010.

[160] Lefteris Sidirourgos, Martin L. Kersten, and Peter A. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *CIDR*, 2011.

[161] Abraham Silberschatz, Henry F Korth, Shashank Sudarshan, et al. *Database system concepts*. 1997.

[162] John Skilling. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2006.

[163] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *TPAM*, 2000.

[164] Asma Souihli and Pierre Senellart. Optimizing approximations of dnf query lineage in probabilistic xml. In *ICDE*, 2013.

[165] Ion Stoica. For big data, moore's law means better decisions. http://www.tableausoftware.com/.

[166] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with. *PVLDB*, 2014.

[167] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *PVLDB*, 2013.

[168] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, 2009.

[169] Antonio Torralba, Robert Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *TPAM*, 2008.

[170] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 2015.

[171] Jiannan Wang, Sanjay Krishnan, Michael J Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, 2014.

[172] Larry Wasserman. *All of Nonparametric Statistics*. Springer, 2006.

[173] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, 1998.

[174] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS*, 2009.

[175] Hadley Wickham. Bin-summarise-smooth: a framework for visualising large data. Technical report, had.co.nz, 2013.

[176] Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, 2000.

[177] Sai Wu, Beng Chin Ooi, and Kian-Lee Tan. Continuous sampling for online aggregation over multiple queries. In *SIGMOD*, 2010.

[178] Fei Xu, Christopher Jermaine, and Alin Dobra. Confidence bounds for sampling-based group by estimates. *TODS*, 2008.

[179] Hao Xu, Jingdong Wang, Zhu Li, Gang Zeng, Shipeng Li, and Nenghai Yu. Complementary hashing for approximate nearest neighbor search. In *ICCV*, 2011.

[180] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and HV Jagadish. Indexing the distance: An efficient method to knn processing. In *VLDB*, 2001.

[181] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[182] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *USENIX HotCloud 2010*, 2010.

[183] Jeremy Zawodny. Redis: Lightweight key/value store that goes the extra mile. *Linux Magazine*, 2009.

[184] Kai Zeng, Sameer Agarwal, Ankur Dave, Michael Armbrust, and Ion Stoica. G-OLA: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, 2015.

[185] Kai Zeng, Sameer Agarwal, and Ion Stoica. iolap: Managing uncertainty for efficient incremental olap. 2016.

[186] Kai Zeng, Shi Gao, Jiaqi Gu, Barzan Mozafari, and Carlo Zaniolo. Abs: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, 2014.

[187] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.

[188] Hao Zhang, Alexander C Berg, Michael Maire, and Jitendra Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*, 2006.

[189] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Ubiquitous computing*, 2008.