Database Learning: Toward a Database that Becomes Smarter Every Time

Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, Barzan Mozafari
University of Michigan, Ann Arbor
{pyongjoo,tajik,michjc,mozafari}@umich.edu

1. INTRODUCTION

In traditional databases, past queries are rarely useful for speeding up future queries. Besides a few limited benefits (see related work below), **the computation performed for answering past queries is often wasted**. However, in an AQP context (e.g., in BlinkDB [1]), there are ample opportunities to re-use previous computations. This is due to the fact that the answer to a query always reveals some *fuzzy knowledge* about the answer to another one, even if the two queries access different tuples and columns. This is because the answers to both queries come from the same underlying distribution which has produced the entire dataset. In other words, **each answer reveals a piece of information about this unknown underlying distribution**. This has long been the essence of machine learning, where past observations are used to improve future predictions.

Our key goal in this paper is to apply the same principle but in a query processing setting, which we call *database learning (DBL)*. To pursue this idea, in a nutshell, we treat approximate answers to past queries as observations, and update our belief on the underlying data based on these observations. Statistically, this update process corresponds to obtaining a posterior probability distribution by conditioning on observations, and we refer to database learning's current understanding of underlying data as *model*. In system perspective, database learning works as a middle-ware between exploratory data analysts and a sample-based approximate query processor [1, 5], as depicted in Figure 1, and improves sample-based answers based on its model. In other words, database learning adds *learning ability* to conventional *memory-less* approximate database systems.[1]

2. RELATED WORK

In traditional databases, the work performed for answering past queries is entirely wasted (i.e., does not benefit future queries), except in a few limited cases.

1. *Adaptive indexing and view selection*: in predictable workloads, columns and expressions commonly used by past queries provide hints on which indices [4, 6, 7] or materialized views [2] to build;
2. *Caching*: the recently accessed tuples will be in memory when future queries access the same tuples.

Adaptive indexing schemes (e.g., database cracking [6]) incrementally refine indices as queries arrive, on demand. However, there is still an exponential number of possible column-sets that can be indexed. Similarly, materialized views are only beneficial when new queries are completely *contained* in the precomputed blocks of data. Database learning is fundamentally different and superior for several reasons:

1. Unlike indices and materialized views, database learning incurs **little storage overhead** as it only retains past N queries and their *aggregate* answers.
2. While indices and materialized views grow in size when the data grows, database learning is **oblivious to the data size** as it only stores final aggregate answers.
3. While existing approaches are effective only when new queries touch prepared areas, database learning can benefit future queries even when their tuples are not contained in previous queries. This is because the trained probabilistic model **spans the entire data** (see Figure 2)

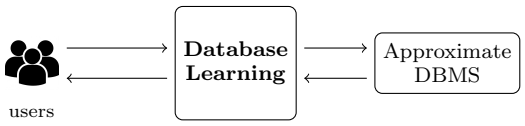


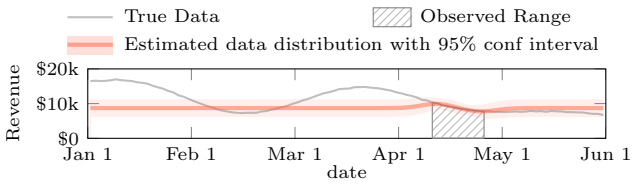
Figure 1: Our proposed system (database learning) is placed between data analysts and an approximate DBMS, and improves the answers by the approximate DBMS.

3. OUR APPROACH

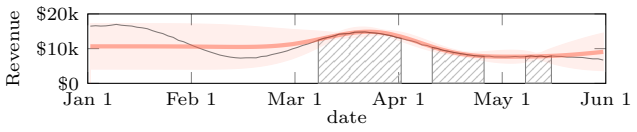
Similar to many learning agents in AI research, the internal model of database learning captures the *most-likely* data characteristics based on observed pairs of query and its (approximate) answer. Database learning relies on the model to produce probabilistic answers to important subsets of analytic SQL queries. The *learning* nature of database learning makes the system most suitable for building an adaptive (and intelligent) system. More specifically, database learning is temporally incremental — the model becomes more sophisticated as more queries are processed — and spatially adaptive — the model accurately reflects a certain area as the query access pattern shifts to that area. Database learning is most appealing in situations in which fast approximate answers are preferred over slower exact answers, since it offers significant speed/accuracy benefits over basic sample-based approximate query processors.

The database learning's novel model-driven query answering mechanism is made possible due to the following contributions of this work: important query statistics (means and covariances between query answers) derived from a random

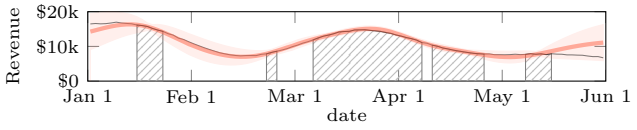
[1]Database learning works with any approximate database systems that returns error-accompanied answers, but our discussion will primarily focus on sample-based approximate database systems.



(a) After two queries. Incorrect confidence interval due to lack of evidence; we have a safeguard mechanism for such cases.)



(b) After five queries. Data characteristics were well-learned.



(c) After ten queries. Accurate for most of areas.

Figure 2: Example of internal model updates.

tuple generation model, and an efficient inference mechanism for computing new query answers using a joint probability distribution function over query answers. Here, the joint probability distribution function over query answers is determined based on the query statistics using a simple but powerful statistical principle called *the principle of maximum entropy* [8]. This principle determines the most-likely explanation of aggregate values given limited amount of knowledge extracted from past queries, which may include the queries with complex conditional expressions in their aggregate functions, or the queries with multiple arbitrary columns in their filtering predicates or group-by statements. After all, this model is used to enhance the accuracies of sample-based approximate aggregate answers at the cost of negligible runtime overheads.

Figure 2 illustrates that how database learning updates its probabilistic model upon processing of ad-hoc queries. Note that, although we depict queries on a one-dimensional space for simplicity, database learning can handle much more complex tables. Due to the model's probabilistic nature, the model is accompanied by confidence intervals; we drew 95% confidence intervals using shaded areas in the figure with the model's best estimations at their centers. For reference, we also accompanied true aggregate values which were computed by issuing large number of queries with very small non-overlapping ranges in their filters. In Figure 2(a) when only two available queries are given, database learning has insufficient amount of information to describe the entire data. However, as more queries are available (Figures 2(b) and 2(b)), database learning's internal model learning's internal model tells a convincing explanation of underlying data even though the past queries overlapped in arbitrary and complex patterns and some of the areas were even untapped by all of those queries. Of course, database learning may not exactly coincide with the ground-truth values for the entire area; however, this is an expected outcome stemming from the adaptive behavior of database learning.

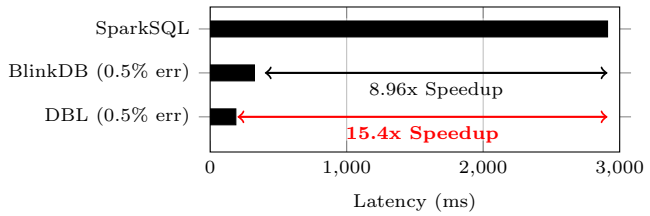


Figure 3: Query processing speedup of database learning (DBL) compared to SparkSQL and BlinkDB. For BlinkDB and database learning, we report their runtime for 0.5% relative error.

4. EXPERIMENTS

We have implemented database learning on top of Spark SQL [3]; database learning modified regular approximate query answers computed by Spark SQL using different sizes of samples. We refer to the sample-based system as BlinkDB, whereas we refer to the exact answer computed based on the original dataset simply as SparkSQL. For performance comparison, we used a TPC-H 100G dataset with query no. 6; multiple queries were generated using different parameter values. Database learning used the first 20 queries to learn its internal model. Then, using the next *unseen* 100 queries, we compared SparkSQL, BlinkDB, and our system (database learning, or DBL). For this experiment, we used 20 recent generation EC2 instances as Spark's worker nodes and cached all datasets in memory before running queries.

Figure 3 compares the query latencies of three systems (SparkSQL, BlinkDB, and DBL). For BlinkDB and DBL, we used a respective sample size that produce the average of 0.5% relative error. Since DBL uses the computations from the past, the sample used by DBL was much smaller than BlinkDB. After all, database learning showed more than 15x speedup compared to SparkSQL and about 1.8 time speedup compared to BlinkDB, respectively.

5. REFERENCES

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB*, 2000.

[3] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. In *SIGMOD*, 2015.

[4] G. Graefe and H. Kuno. Adaptive indexing for relational keys. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 69–74, March 2010.

[5] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[6] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *CIDR*, 2007.

[7] E. Petraki, S. Idreos, and S. Manegold. Holistic indexing in main-memory column-stores. In *SIGMOD*, pages 1153–1166, 2015.

[8] J. Skilling. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2006.