Bayesian AutoML for Databases via the InferenceQL Probabilistic Programming System

Ulrich Schaechtle¹ Cameron Freer² Zane Shelby¹ Feras Saad² Vikash Mansinghka²

¹Digital Garage
²Massachusetts Institute of Technology

Abstract InferenceQL is a probabilistic programming system for scalable Bayesian AutoML from database tables. InferenceQL is designed to help make Bayesian approaches to data analysis more accessible to broad audiences and to assist experts in auditing and improving the quality of data, models, and inferences. Unlike traditional probabilistic programming systems, InferenceQL provides automation for learning models online using nonparametric Bayesian structure learning of probabilistic programs. Experts can override these models with custom probabilistic programs for specific subsets of variables and conditional distributions. For a broad class of models, InferenceQL can generate realistic synthetic data subject to constraints and can automatically compute exact probabilities and mutual information values. Finally, InferenceQL aims to enable scalable Bayesian model criticism via posterior predictive checks, data quality screening via conditional probability calculation, fairness auditing via conditional probability ratios, and synthetic data generation to enhance privacy. These capabilities are accomplished using constructs that interleave standard database queries with Bayesian inference.

Automated Bayesian inference from databases is important and useful in several ways. First, many real-world databases have high rates of missing values, more fields than observed records, heterogeneous data types, high rates of data entry error, and other factors that complicate the application of traditional ML-based AutoML techniques [9, App. E]. Second, many real-world applications benefit from uncertainty quantification, interactive model checking and model criticism, and conditional probability estimation for ad-hoc fairness auditing. These problems are naturally formulated in terms of Bayesian inference [11, 28].

InferenceQL is a probabilistic programming system for automated Bayesian inference from database tables. InferenceQL provides a domain-general mechanism for online Bayesian structure learning of probabilistic program source code, as well as domain-general mechanisms for scalable exact and approximate inference in these probabilistic programs. Users thus do not have to know how to write probabilistic programs in order to use InferenceQL to solve problems. Instead, users rely on automated data modeling techniques to navigate the design choices that might otherwise be handled by experienced modelers. InferenceQL also enables Bayesian inference operations to be interleaved with ordinary SQL operations, yielding complex database-native workflows for Bayesian AutoML. InferenceQL has been used successfully in field tests for a broad range of applications, including AutoML for clinical trial oversight in three real-world clinical trials.

This workshop paper introduces InferenceQL via an exploratory data analysis application. It also briefly reviews the system architecture of InferenceQL and the class of probabilistic programs that deliver its AutoML capabilities. It presents preliminary quantitative results from experiments comparing InferenceQL’s modeling accuracy to GLM, VAE, and CTGAN baselines. Finally, it reviews related work, including both modeling formalisms and ML and database integrations, and discusses some limitations and broader impacts.
The InferenceQL system automates data analysis and machine learning tasks by allowing users to input data tables and queries and to automatically generate answers for them (Figure 1). It consists of a probabilistic program synthesis component [27] that creates generative model programs that are called row generators. The InferenceQL query planner and query engine use row generators to answer questions about the data and the domain by querying an underlying probabilistic model.

Figure 2 shows an example of probabilistic program synthesis, which takes a heterogeneously-typed data table of satellites (maintained by the Union of Concerned Scientists [32]) and returns a probabilistic program that models the data. Figure 3(a) shows the high-level interface to creating synthesizing programs. Users can then compare synthetic data (generated from the probabilistic programs) with observed data in order to develop intuition about what the model learned from the data, shown in Figure 3(b). InferenceQL can generate virtual data from both marginal distributions and conditional distributions given a user-specified predicate (code box in Figure 3(b)). The two plots in Figure 3(b) illustrate a qualitative goodness-of-fit in the sense that distribution of synthetic (orange dots) samples appears to approximately match the observed data (black dots).

Data analysts can use the query language to search for probable anomalies and data-entry errors, shown in Figure 3(c). To find values for the column Period\_minutes that the model considers

1 Example

The InferenceQL system automates data analysis and machine learning tasks by allowing users to input data tables and queries and to automatically generate answers for them (Figure 1). It consists of a probabilistic program synthesis component [27] that creates generative model programs that are called row generators. The InferenceQL query planner and query engine use row generators to answer questions about the data and the domain by querying an underlying probabilistic model.

Figure 2 shows an example of probabilistic program synthesis, which takes a heterogeneously-typed data table of satellites (maintained by the Union of Concerned Scientists [32]) and returns a probabilistic program that models the data. Figure 3(a) shows the high-level interface to creating synthesizing programs. Users can then compare synthetic data (generated from the probabilistic programs) with observed data in order to develop intuition about what the model learned from the data, shown in Figure 3(b). InferenceQL can generate virtual data from both marginal distributions and conditional distributions given a user-specified predicate (code box in Figure 3(b)). The two plots in Figure 3(b) illustrate a qualitative goodness-of-fit in the sense that distribution of synthetic (orange dots) samples appears to approximately match the observed data (black dots).

Data analysts can use the query language to search for probable anomalies and data-entry errors, shown in Figure 3(c). To find values for the column Period\_minutes that the model considers
improbable in light of the data, the query (left code box) produces the result by comparing the probability of the value for Period_minutes marginally and conditionally in the WHERE clause. The only rows returned are those whose conditional probability is lower than the marginal; the probabilistic program beats the default (a further 300x improvement in RMSE).

Finally, users with domain expertise can customize probabilistic programs. Figure 3(d) shows an example custom orbital model from physics. To quantitatively assess the goodness-of-fit, we first split the data into training and test data and build three models: the automatically synthesized default model, a generalized linear model (GLM), and a custom probabilistic program for noisy orbital physics. We then predict a column in the held-out data set. The default model predicts more accurately than the GLM (4x more accurate via root mean square error (RMSE)) and the custom probabilistic program beats the default (a further 300x improvement in RMSE).

Figure 3: A representative data analysis workflow in InferenceQL on the satellites data.
2 Experiments

We now report experiments evaluating InferenceQL against statistical and neural baselines.

Virtual data benchmark. Table 1 shows the average Jensen-Shannon divergence between virtual and observed data for all pairwise marginals in 10 datasets from the UCI machine learning repository [8], according to simulations from InferenceQL, Gaussian copulas [23], CTGAN [34], and TVAE [34]. The bold entries indicate statistically significant lowest error under a Bonferroni corrected Mann-Whitney *U* test, which are achieved by InferenceQL in 8 of 10 benchmark problems and zero times by other techniques.

Anomaly detection benchmark. Table 2 shows that InferenceQL detects a higher percentage of anomalies than does a GLM baseline on three datasets from the UCI repository. Anomalies were inserted into a target column by flipping the class label in each row with probability 0.05 and detected using a query similar to the one in Figure 3(d).

Query optimization. The InferenceQL query planner contains a built-in optimization for querying row generators specified in the SPPL language [28]. Table 3 shows the runtime of InferenceQL queries for computing the conditional probability of all cell values in one target column given all the other values in the same row, for 10 datasets from the UCI repository. The third column shows the runtime using InferenceQL’s independence analysis optimization, which statically eliminates from the query all conditioning variables that are structurally independent of the target variable. The fourth column shows the runtime using InferenceQL without independence analysis and the final column shows the runtime using the Python API to SPPL, which both do not automatically leverage independence analysis and are slower in cases where independencies can be exploited.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Target</th>
<th>InferenceQL (SPPL backend)</th>
<th>Python API (SPPL)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Independence Analysis</td>
<td>Default Optimization</td>
</tr>
<tr>
<td>Nursery</td>
<td>Evaluation</td>
<td>11.14 ± 7.31</td>
<td>501.35 ± 571.08</td>
</tr>
<tr>
<td>Tumor</td>
<td>Type</td>
<td>1.99 ± 0.34</td>
<td>3.21 ± 0.51</td>
</tr>
<tr>
<td>Flare</td>
<td>Num_common_flares</td>
<td>6.96 ± 2.56</td>
<td>14.32 ± 14.27</td>
</tr>
<tr>
<td>Car</td>
<td>Evaluation</td>
<td>13.03 ± 4.69</td>
<td>153.91 ± 275.4</td>
</tr>
<tr>
<td>Mushroom</td>
<td>Edible?</td>
<td>31.34 ± 6.17</td>
<td>34.07 ± 6.87</td>
</tr>
<tr>
<td>Soybean</td>
<td>Disease</td>
<td>9.44 ± 3.05</td>
<td>11.7 ± 2.26</td>
</tr>
<tr>
<td>Breast-cancer</td>
<td>Diagnosis</td>
<td>5.07 ± 0.71</td>
<td>6.78 ± 0.73</td>
</tr>
<tr>
<td>Heart-disease</td>
<td>Present?</td>
<td>3.49 ± 1.49</td>
<td>11.61 ± 8.11</td>
</tr>
<tr>
<td>Connect-4</td>
<td>White_can_win</td>
<td>34.24 ± 24.61</td>
<td>65.26 ± 59.99</td>
</tr>
<tr>
<td>Chess</td>
<td>Outcome</td>
<td>61.83 ± 45.34</td>
<td>86.27 ± 51.3</td>
</tr>
</tbody>
</table>
3 Related Work

Various AutoML systems have been developed for tabular data, many of which focus on solving a specific discriminative problem (e.g., regression or classification); whereas InferenceQL the user automatically obtains a generative model that can be queried repeatedly to answer a wide range of questions about the data. These include Amazon’s AutoGluon-Tabular [9] and SageMaker Autopilot [7], Google Cloud Platform AutoML Tables [17], Uber’s Ludwig [20], H2O AutoML [16], and a number of earlier systems such as Auto-WEKA [31], auto-sklearn [10], hyperopt-sklearn [2], TPOT [21], autoxgboost [30], ML-Plan [19], OBOE [35], GAMA [12], and Auto-Keras [15]. A survey and comparison of many of these systems can be found in Erickson et al. [9, §3].

InferenceQL is most similar to the BayesDB [18] probabilistic programming system. Like InferenceQL, BayesDB requires the underlying models to satisfy the composable generate population model interface [24]. InferenceQL aims to support new queries that include predicates as constraints and custom models in the Gen probabilistic programming language [5].

InferenceQL makes use of sum-product expressions [28], which are a type of probabilistic circuit [6]. These symbolic representations are designed to balance modeling expressiveness with tractable inference. Previous database systems have incorporated probabilistic circuit representations to speed up queries [13, 22, 33] outside of a probabilistic programming context.

Probabilistic databases [29] are a class of systems that take a different approach from InferenceQL, where they typically assign weights to facts in a database and implement queries that compute the probability of Boolean formulas. Unlike InferenceQL, standard probabilistic databases do not include automatic probabilistic model discovery from high-dimensional and heterogeneously-typed data tables. Another probabilistic approach to database systems and query languages involves augmenting relational algebras with techniques that enable predictive modeling, learning, and inference. Relational algebras have been extended by functions for imputation [4], time series prediction [1], random data generation [14] and simulation [3].

4 Conclusion

Limitations. A central limitation of InferenceQL is the lack of large-scale evaluations of its accuracy and performance that span a range of different data regimes. In principle, InferenceQL should be able to match the accuracy of many real-world ML deployments by using hybrids of generative models with decision-tree classifiers (as implemented in its SPPL backend), though it remains to be seen how effective and competitive with other AutoML approaches this approach is in practice. Also, once users leave the fully-automated setting (e.g., to customize or override the learned probabilistic programs), the InferenceQL API for custom models as in Figure 3(d) can be challenging to implement correctly. Finally, InferenceQL only applies to cross-sectional data tables; it would be worthwhile to extend InferenceQL with domain-general Bayesian structure learning methods for multivariate time series [25] or relational systems [26].

Broader Impact. One potential benefit of InferenceQL is educational. If it turns out (much like SQL) to be learnable by many end-users, then InferenceQL has the potential to increase access to and literacy in Bayesian reasoning for a broad audience. InferenceQL also has the potential to reduce the marginal cost of Bayesian data analysis for applied practitioners, which drives both beneficial impact and risks of harm. Benefits include improved analytics and decision-making capabilities in the public interest, especially in applications where real-time learning is valuable, such as adaptive design of scientific experiments. The risks include reduced cost for invasion of privacy, as well as analytics and decision-making towards other harmful aims.
References


