Integrating a Fast, Non-parametric Bayesian Inference Engine into BayesDB

Michael Truell

September 2017

Abstract

Loom is a non-parametric Bayesian inference engine that finds meaning in large datasets. We've integrated the inference abilities of Loom into the BayesDB probabilistic programming platform, in order to allow non-statisticians to utilize the Loom algorithm for their research. Second, we have experimentally validated the ability of the now-complete Loom algorithm. We have shown that Loom vastly improves on the scalability of its predecessor, the CrossCat machine learning algorithm, with the ability to analyze datasets with 1,000,000 rows in a matter of hours. Loom matched the performance of an expert statistical analysis on a diabetic care dataset in a few hours, with no hand-tuning of parameters, and using just 15 lines of source code.

1 Introduction

The analysis of complex, high-dimensional, and sparse databases is a common statistical and scientific problem. To borrow an example from computational biology, the LIGAND enzymatic database [1] is made up hundreds of thousands of rows of data, each containing detailed metrics about the three-dimensional structure and interactions of one of ten thousand enzymes. Meaning is hiding in this vast, tabular list of numbers; encoded in it are the fundamental interactions that allow human bodies to function. Statistical analysis tools attempt to tease out meaning in datasets like LIGAND.

One example of a way in which statistical inference systems extract meaning from data is through the discovery of dependent relations between variables in a database. For example, a perfect statistical modeling system would, given a database of economic indicators, be able to self-discover that national GDP growth is partially dependent on federal minimum wage policy. The relationship between the two is non-trivial and masked by lots of statistical noise, but a signal is hiding in the data. Another example of a candidate inference query is the prediction of new data points. Returning to our economics example, an ideal statistical system should, given a profile of the economy of a country, be able to estimate the country's unemployment rate.

Complex statistical inferences, such as dependency detection and data prediction, have recently become a topic of fervent research [2, 3, 4]. However, they remain statistically and computationally difficult. First, real-world datasets may contain anomalous or missing values. As such, they require cleaning and interpolation before pairwise statistics can be calculated or computational analysis can begin. Second, the modeling

assumptions that underlie standard hypothesis testing techniques, such as R^2 and HSIC, are often not appropriate, because real-world relationships are nonlinear and multivariate. Third, as the number of database variables grows, it becomes harder to detect true relationships while suppressing false positives.

The MIT Probcomp Group [5] has proposed a promising approach to statistical inference on sparse and complex datasets that combines probabilistic programming, information theory, and non-parametric Bayesian machine learning algorithms. This system uses the group's Bayesian learning model called CrossCat [6] to group related rows and columns in a database according to a hierarchy of probabilistic mixture models. Using these groupings, CrossCat can model the distribution of each variable in the database. The system is included and wrapped in BayesDB and BQL [7], an SQL-like database and programming language that allow for the easy use of data science utilities.

The CrossCat machine learning system has shown improved sensitivity and specificity over traditional correlation modeling techniques, such as R^2 and HSIC, and even state-of-the-art machine learning systems [5]. However, CrossCat only functions on small to medium sized datasets, often those containing less than 10,000 rows and 200 columns. CrossCat's performance issues stem from both implementation inefficiency and algorithmic latency.

To address these issues, Dr. Fritz Obermeyer of UC Berkely EECS began and spearheaded the development of Loom, a new machine learning algorithm, with his publishing of the subsampling annealing improvement to the CrossCat algorithm [8]. In addition to mitigating CrossCat's poor scaling ability through the implementation of subsample annealing, Loom further improves upon CrossCat with a parallel, low-level software implementation.

We've integrated the Loom engine with BayesDB, an easy to use SQL-like database platform, which is amenable to non-computer-scientists. This integration vastly broadens the scope of Loom, converting it from a messy jumble of source files to a general-purpose statistical tool. We have benchmarked Loom's inference against its CrossCat predecessor. Finally, to showcase Loom's speed, effectiveness, and ease-of-use, we have applied the system to the analysis of a large-scale diabetic patient dataset, where it matched the performance of an expert statistical analysis in hours and using just 15 lines of source code.

2 Background

In this section, we present an introduction to simple probabilistic mixture modeling and the Loom machine learning algorithm.

2.1 Mixture Models

A mixture model is a machine learning method for unsupervised classification. A mixture model fits a set of finite random distributions to a distribution of real-life data. Each distribution represents a hypothesized subpopulation, or class, of the data. One common example of a mixture model is the Gaussian mixture model (shown in Figure 1), which fits a constant number of Gaussian distributions to a distribution of data using Expectation Maximization, a simple hill-climbing optimization method.

A DPMM is a non-parametric extension of the concept of a mixture model. In practice, the number of classes appropriate for a real-world dataset is unknown. Thus, DPMMs function by maintaining an infinite distribution over a set of candidate mixture models, each with a different number of classes, and therefore

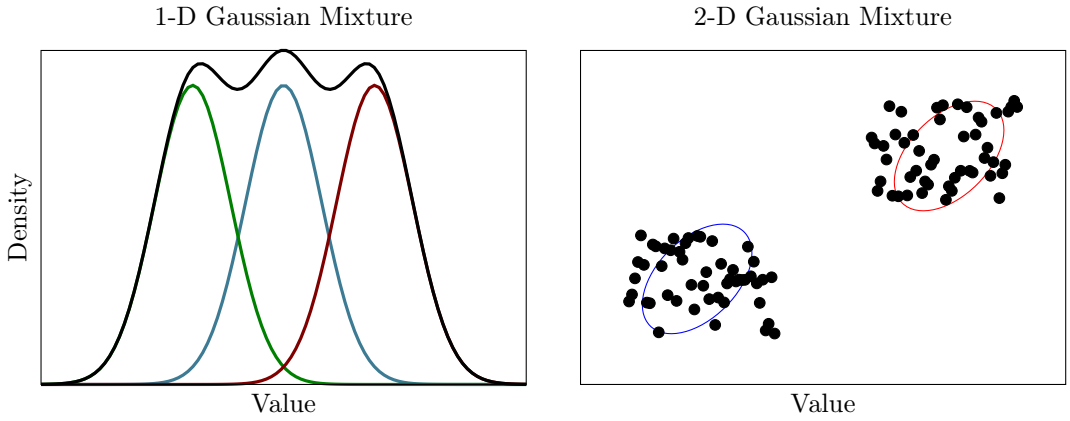

Figure 1: Left: A simple univariate Gaussian mixture using three constituent Gaussian distributions (green, blue, red), each representing a subpopulation of the data, to model a probability distribution (black). **Right**: An example of a bivariate mixture model. Two individual univariate Gaussian mixture are combined to model the joint distribution of a pair of variables.

constituent distributions, that starts from 1 and goes to infinity. DPMMs can be used to model many statistical types of data. For example, they may be paired with Gaussian mixtures to model real numbers, Gamma-Poisson mixtures to model integers, or Dirichlet-Discrete mixtures to model categorical data.

2.2 Loom

Loom is a machine learning algorithm for clustering heterogeneous, sparse datasets [8]. Loom employs a hierarchy of DPMMs. One DPMM is used to partition the columns of a datset into clusters. The daughter mixtures of this column DPMMs are themselves DPMMs, and each partitions the rows of the dataset within a specific column cluster. The daughter mixtures of row DPMMs are simple mixture models, with the type of the mixture changing according to the statistical type of the data being modeled.

A visualization of a clustering of a hypothetical dataset of animal characteristics is shown in Figure 2. The groupings that Loom's predecessor infers have been shown to be consistent with accepted findings and common-sense knowledge in multiple domains [6].

2.3 BayesDB

The BayesDB project aims to allow non-computer scientists to easily query the implications of their data and perform inferences as easily as querying the data itself [7]. BayesDB effectively serves as a common, reusable interface between machine learning models, such as the MIT CrossCat model, and data storage. The project uses its own SQL-like programming language called BQL. BayesDB is currently used by a small, closed group of NGOs, governmental departments, and academic labs. In just days, BayesDB users have bested data scientists on benchmarks such as flu data analysis and weather modeling.

Figure 2: Figure from [6]. **Top**: A visualization of a CrossCat/Loom clustering on an example dataset of animal characteristics. Animal names are on the y-axis, and animal features are on the x-axis. A green point indicates that the animal at said y coordinate does have the feature on said x coordinate; for example, the square at the top left of the figure indicates that an "Alligator" has the "bites" characteristic. **Bottom**: A visualization of Loom's learned clusters on the animal dataset. The algorithm has partitioned the features (the "columns" or "labels" of the dataset) into three, large related blocks (A-C). Groups of related animals within each of these three blocks are denoted by black bold lines.

3 Materials and Methods

We've integrated Loom's inference queries with BayesDB/BQL programming language frontend and added a few inference queries on top of the Loom system.

3.1 Integrated Inference Abilities

We enumerate Loom's suite of inference abilities below. Each inference query operates on tabular data – data in the form of rows and columns. Columns may be viewed the axes of a data set and rows as the actual data points.

3.1.1 Detecting Dependencies Between Columns

Algorithm 1 Dependency Detection

Require: Loom models L_n for $n = 1, \ldots, N$; Columns c_1 and c_2;
Ensure: A normalized column dependency value for c_1 and c_2 across all L

$s \leftarrow 0$
for $n = 1, \ldots, N$ **do**
    $k_1 \leftarrow \text{GetColumnCluster}(L_n, c_1)$
    $k_2 \leftarrow \text{GetColumnCluster}(L_n, c_2)$
    if $k_1 == k_2$ **then**
        $s \leftarrow s + 1$
return s / N

To detect dependencies between a pair of columns from a dataset, it is natural to use a Monte Carlo estimate of the probability that the two columns were grouped into the same Loom cluster (Algorithm 1) [6]. To calculate this, we may sample a set of Loom models. Each model will have a uniquely seeded pseudo-random-number-generator and will construct a different set of clusters.

3.1.2 Estimating Row Similarity

Algorithm 2 Row Similarity

Require: Loom models L_n for $n = 1, \ldots, N$; Rows r_1 and r_2;
Ensure: A normalized row similarity value for r_1 and r_2 across all L

$s \leftarrow 0$
for $n = 1, \ldots, N$ **do**
    $C \leftarrow \text{GetAllColumnClusters}(L_n)$
    $s_n \leftarrow 0$
    for $C_i \subset C$ **do**
        $k_1 \leftarrow \text{GetRowCluster}(L_n, C_i, r_1)$
        $k_2 \leftarrow \text{GetRowCluster}(L_n, C_i, r_2)$
        if $k_1 == k_2$ **then**
            $s_n \leftarrow s_n + 1$
    $s \leftarrow s + s_n/\text{Len}(C)$
return s / N

Exploratory analyses often make use of a "similarity" function defined over a pair of rows. Such a function quantifies the distance between two rows in a dataset. In the mixture model literature, the similarity problem is often formalized as the probability that two rows were generated from the same underlying statistical

model. The Loom similarity function may be seen as roughly equivalent to the column dependency function above, but with the important distinction of operating on rows instead of columns (Algorithm 2) [6].

3.1.3 Predicting New Data Points

Loom was given the ability to predict new row values, ranging from the interpolation of a singular missing value to the simulation of whole new rows of data [6]. Formally, Loom must calculate a set of missing row values given a set of conditional row values. This prediction can be done in two, high-level steps. First, for each column cluster of each Loom model, we must estimate the row cluster to which our incomplete row of conditional values belongs. To determine this, we can sample the conditional density of Loom's outermost DPMM. We must then simulate our missing row values. To do so, we sample from the conditional density of the inner DPMM for our inferred row cluster.

3.1.4 Ranking Rows By Relevance

Algorithm 3 Row Similarity

Require: Loom models L_n for $n = 1, \ldots, N$; Candidate row r; Query row q; Column of interest c;
Ensure: A normalized measure of the relevance of r in formulating predictions about q in the c column across all L

```
s ← 0
for n = 1, ..., N do
    cc ← GetColumnCluster(L_n, c)
    rc ← GetRowCluster(L_n, cc, r)
    qc ← GetRowCluster(L_n, cc, q)
    if rc == qc then
        s ← s + 1
return s / N
```

Predictive relevance is a candidate search ranking function for tabular data. The predictive relevance of row r to a query row q in the context of column c measures the relevance of r in formulating predictions about q in the c column. From an information theoretic standpoint, predictive relevance is equivalent to the posterior probability that the mutual information between r and q in the context of c is non-zero.

Predictive relevance may be elegantly derived from Loom's groupings (Algorithm 3) [9]. A row cluster is denoted by R and is indexed by n, the index of a single Loom model. To compute predictive relevance, we count and normalize the number of row clusters that r and q share across models in the column cluster of column c, like so:

3.2 BayesDB Integration

We wrap our inference queries in BayesDB, an easy to use SQL-like database. The accessibility of BayesDB will allow professionals and scientists to leverage the power of Loom. This will hopefully expand our work's impact to extend past our respective field and the specific applications presented by this paper to many branches of science and industry.

The inference system was fully integrated into the grammar of the Bayesian Query Language, BayesDB's SQL-like interpreted data manipulation language. An example of a full BQL workflow for ingesting data and computing dependency estimations using Loom for the Gapminder economic database is show in Listing 1.

```
-- Ingest CSV file
CREATE TABLE "sql_table" FROM 'resources/gapminder.csv';
-- Create analysis population
CREATE POPULATION "population" FOR "sql_table" WITH SCHEMA (
    GUESS STATTYPES FOR (*);
);
-- Create analysis schema for gapminder dataset
CREATE ANALYSIS SCHEMA "loom_model" FOR "population" USING loom;
-- Analyze gapminder dataset: i.e. compute dependency values etc
INITIALIZE 1 ANALYSIS FOR "loom_model";
-- Calculate Dependencies
ESTIMATE COLUMN DEPENDENCY FROM PAIRWISE VALUES OF "loom_model";
```

Listing 1: Ingesting the Gapminder dataset [10] and performing dependency detection on its columns in just seven lines of source code. First, the dataset is imported from a csv file. Next, it is converted from a SQL table to a BQL population. Loom is then initialized and is made to perform an analysis of the Gapminder dataset. Finally, we query loom for the dependency values of the columns of this dataset (these are economic indicators, such as gross exports and GDP).

In addition to integration into the BQL grammer, Loom interfaces with MIT Probcomp's existing implementation of CrossCat. Specifically, a BayesDB user is given the ability to convert a Loom model to a CrossCat model after data ingestion and inference, alowing Loom users to use the CGPM inference engine.

4 Results

We have shown that Loom vastly improves on the scalability of the CrossCat machine learning system, rendering analysis of datasets of up to at least 1,000,000 rows feasible. In addition, Loom performs many inference queries faster than the CrossCat system. Finally, Loom's potential was demonstrated on a dataset of diabetic hospitalizations, on which Loom matched the performance of an expert statistical analysis in just 15 lines of code and with no parameter tuning.

4.1 Loom Benchmarking

Loom and CrossCat perform two distinct types of computation. The first is **analysis** on a dataset, where Loom or CrossCat learns its partitioning of a datset. The second is **querying**, when inference queries, such as column dependency detection, are performed. During analysis, Loom or CrossCat fits its mixture models to a dataset and constructs its partition of the rows and columns of said datset. Once analysis has been run, Loom or CrossCat may perform the inference queries enumerated earlier in this paper (predict, dependency, similarity, etc.).

It is important to note that CrossCat and Loom block their analysis in very different ways. CrossCat performs short, iterative passes (or "increments") over a dataset, slightly improving the fit of its DPMMs and the quality of its column and row partitions with each run. However, due to the algorithmic construction of subsample annealing, Loom performs its analysis in one large annealing step. Thus, one Loom analysis is not comparable to one CrossCat increment, making it difficult to compare the analysis runtimes of the two. A rough heuristic is 1 Loom analysis $\approx$ 100-1,000 CrossCat increments.

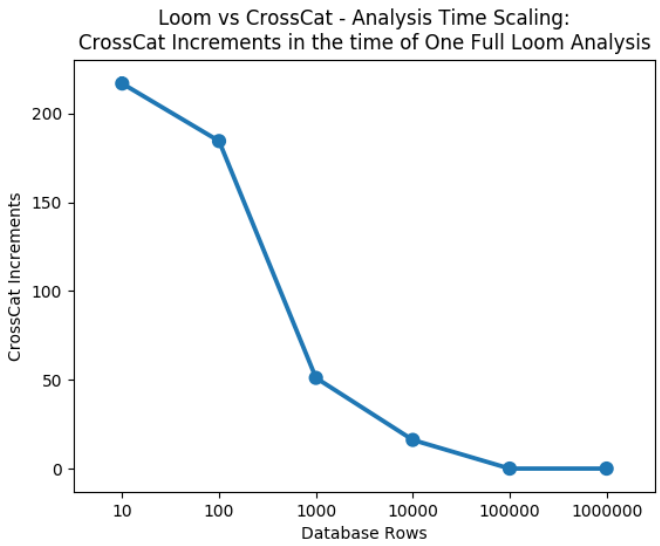

Figure 3: This graph shows the number of analysis increments that CrossCat can perform during the same time as one full Loom analysis. One Loom iteration is roughly comparable to hundreds of CrossCat increments. CrossCat can perform many hundreds of increments on datasets of < 1000 rows. At 1,000 rows, however, CrossCat can only perform 50 increments. On datasets of 10,000 to 1,000,000 rows, CrossCat cannot finish one increment during the time of a complete Loom analysis. All datasets were synthetic and had 32 columns.

The analysis runtimes of Loom and CrossCat are compared in Figure 3. Loom scales vastly better than the CrossCat machine learning algorithm. Loom is able to finish a full analysis of datasets ranging from T rows to 1,000,000 rows before CrossCat could even perform one of its increments. One should note that all Loom analyses completed in less than 48 hours. However, single increments of CrossCat on the one million dataset did not finish within several days and, thus, were never run to completion.

The query speeds of Loom and CrossCat are compared in Figure 4. Loom shows significantly better scaling on big datasets and complex inference queries, though is slower on simple, repeated queries.

4.2 Comparison to Expert Statistical Analysis

To test Loom on in a real-life context, we attempted to replicate a study of national diabetic care [11]. The authors of said study assembled a dataset of 100,000 hospitalizations of diabetic patients from 1999 to 2008. Information recorded about each hospital stay included the patient's current medications, the results of a host of blood tests, demographic information, and future medical incidents. The authors analyzed this dataset to determine the relationship between patient readmit rates and the administration of an HbA1c blood test, a measure of glycated hemoglobin which is used as a proxy for one's three-month average blood plasma glucose concentration.

Strack's analysis required extensive preprocessing and variable control. The authors enlisted a set of "clinical experts" to perform feature selection and prune confounding data entries. They fit a multivariate logit regression model to different partitions of the dataset and controlled for a set of hand-picked variables, including age, primary diagnosis, and the specialty of the admitting doctor. A description of this analysis

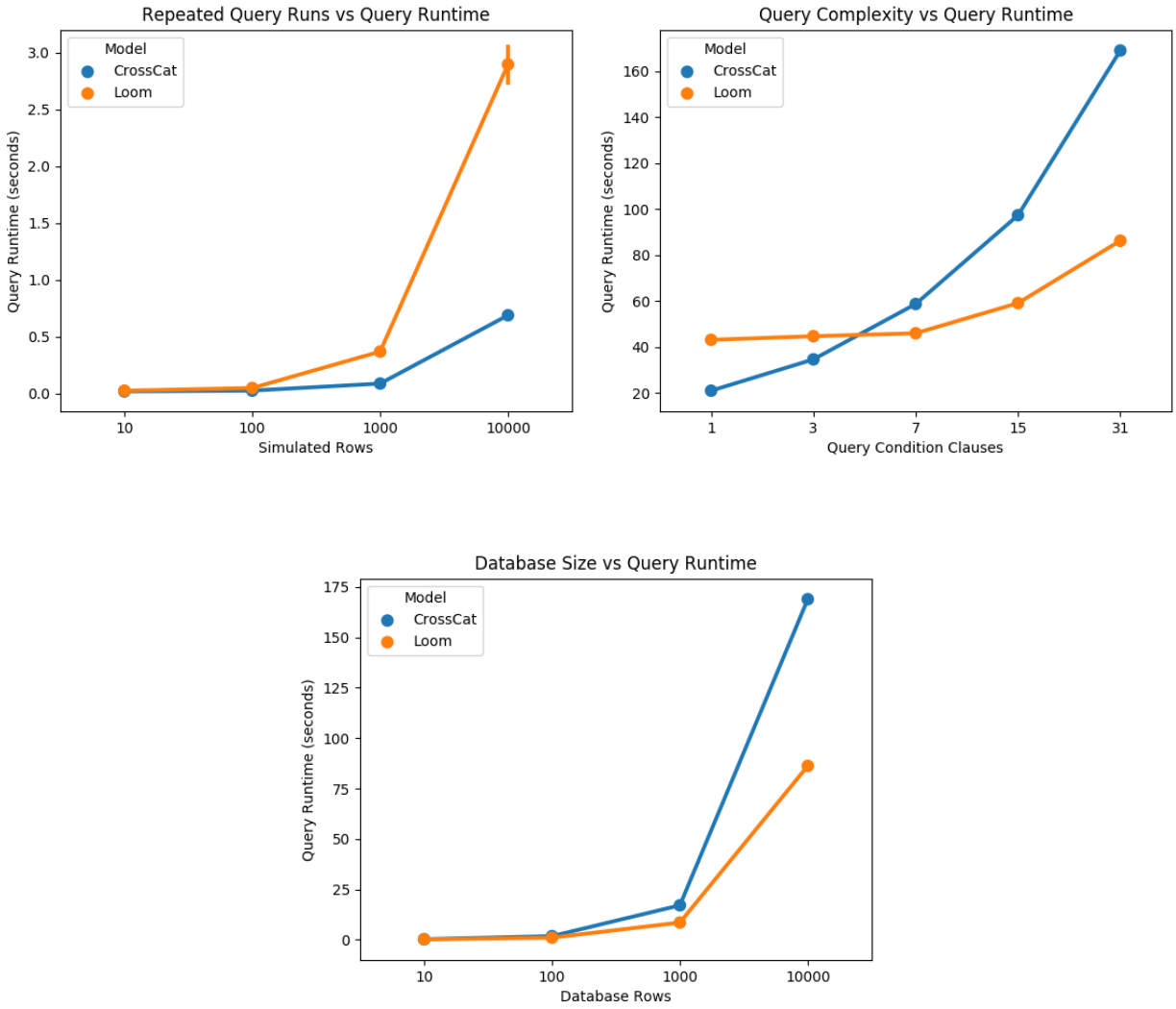

Figure 4: An evaluation of the inference query speed of Loom and CrossCat. Loom shows significantly better scaling (lower runtime) on big datasets and complex inference queries, though is slower on simple, repeated queries. **Top Left**: CrossCat and Loom are repeatedly asketd to predict new dataset rows. **Top Right**: CrossCat and Loom are asked to predict the likelihood of a hypothetical data entry occuring, and the number of prior conditions (aka. the complexity of the query) is varied. **Bottom**: CrossCat and Loom are again asked to predict the likelihood of a hypothetical data entry occuring, and the size of the dataset is varied. All datasets had 32 columns and were synthetically generated.

and its parametrization occupies six, full pages of their paper. In conclusion, they found that, though only administered to 18% of hospitalized diabetics, the HbA1c test significantly lowered early readmission rates, suggesting it should be leveraged much more widely by physicians.

We have applied Loom to the above dataset in order to evaluate the algorithm against an typical data analysis performed by computer scientists and domain experts. Loom requires no preprocessing of the data, no human-tuned parameter selection, and just fifteen lines of BQL source (a listing of this source is included in the appendix). Loom's analysis of the dataset took about six hours for an ensemble of 64 models on a 64 core server.

```
CREATE TABLE "dia" FROM 'resources/diabetes.csv';
CREATE POPULATION "dia_pop" FOR "dia" WITH SCHEMA (
    GUESS STATTYPES FOR (*);
    MODEL
    "diag_1",
    "diag_2",
    "diag_3"
    AS unboundedcategorical;
    IGNORE race, gender, weight, payer_code, patient_nbr;
);
CREATE ANALYSIS SCHEMA "dia_meta" FOR "dia_pop" USING loom;
INITIALIZE 64 ANALYSIS IF NOT EXISTS FOR "dia_meta";
ANALYZE "dia_meta" FOR 1 ITERATIONS WAIT;
ESTIMATE DEPENDENCE PROBABILITY FROM PAIRWISE VARIABLES OF dia_pop;
```

Listing 2: The full dependency analysis source for our analysis of a dataset of diabetic hospitalizations from 1999-2008 [11].

In accordance with the expert analysis [11], Loom found a significant coupling between the administration of an HbA1c test and readmission rates, assigning the pair a dependency value of 0.8413; dependency values range from 0 to 1, with a value of 1 implying full dependence. A full heatmap of dependence values is shown in Figure 5. This result suggests that Loom is competitive with expert, domain-specific statistical analysis, while requiring comparatively little time, and effort.

5 Conclusion

Loom, a Bayesian nonparametric inference engine, was fully integrated into BayesDB. This vastly broadens the scope of the Loom algorithm, taking it from a web of programmatic calls to a general-purpose statistical tool. The scaling ability of the Loom algorithm was benchmarked. Loom was shown to successfully ingest datasets of up to a 1,00,000 rows in under a day, while CrossCat could only scale to datasets of 10,000 rows or fewer. Loom applied a large dataset of diabetic patient hospitalizations. The algorithm matched the performance of an expert statistical analysis in a few hours with no hand-tuning of parameters.

6 Acknowledgments

I would like to thank my mentor Feras Saad, my supervisor Dr. Mansinghka, and the MIT Probcomp group for being welcoming teachers and hosts. I would like to thank Peter Lu for his helpful direction and tutelage.

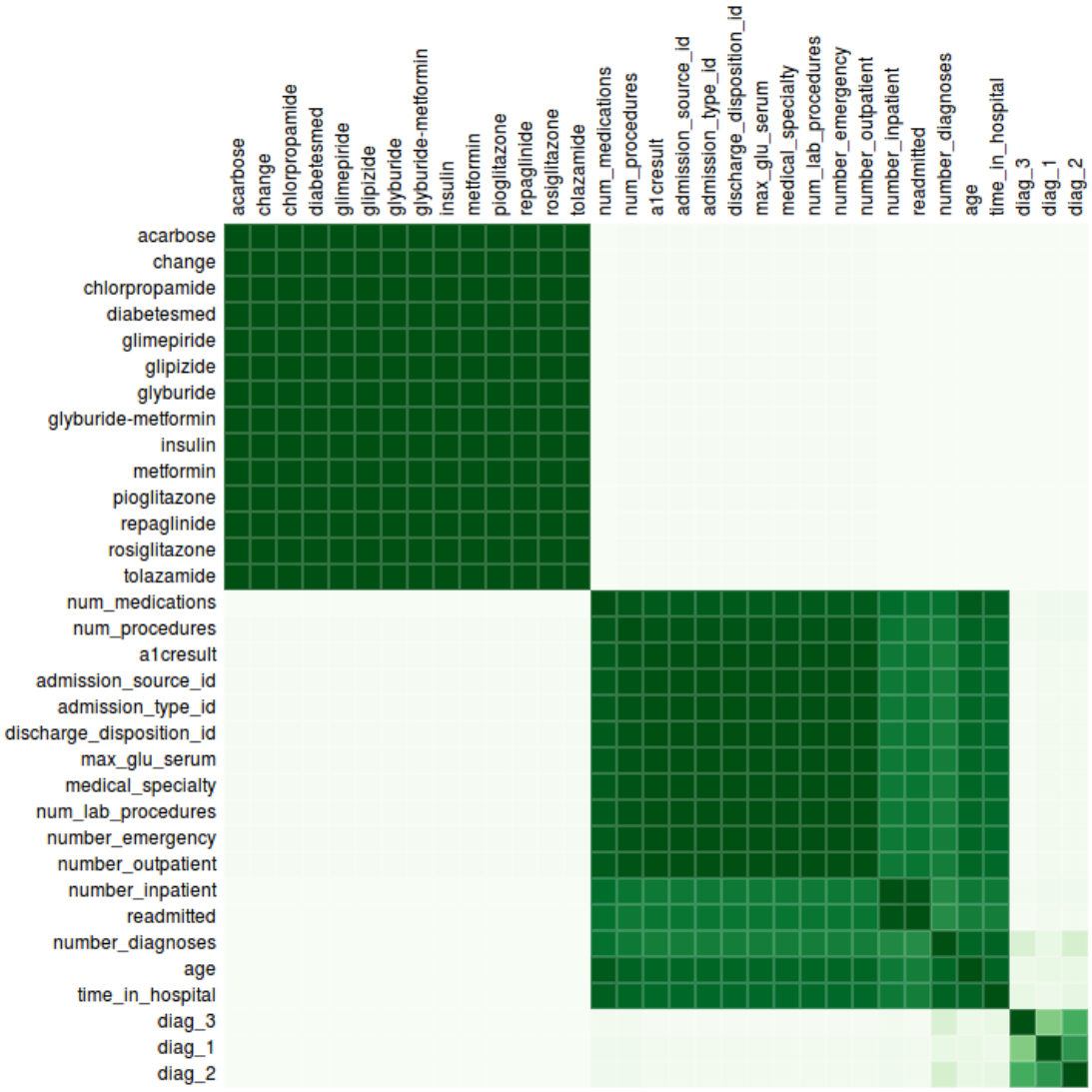

Figure 5: A heat map of the pairwise column dependency values that Loom calculated on the diabetes dataset. Green is dependent, while white is independent. **Top Left**: Loom learns to group all of the medications in one large dependent block. This agrees with our intuition; the medications that one may take are restricted by the set that one is already taking, and medications that target the same condition may be commonly assigned together. **Bottom Right**: Loom also learns a dependent relationship between preliminary (diag_1), secondary (diag_2), and tertiary (diag_3) diagnoses.

I would like to acknowledge the CEE and the RSI for providing me with the opportunity to perform this research and MIT for hosting me. I would like thank my counselors and peers for making RSI an unforgettable experience. Finally, I would like to acknowledge my many sponsors for supporting this research internship, including the Department of Defense, Ansatz Capital, Two Sigma, BEST, Mr. Christopher Chang, Mr. John Yochelson, Andreas Fibig, Dr. Leighton Symons, Dr. Mark Epstein, and Dr. Elyse Rafal.

References

[1] S. Goto, T. Nishioka, and M. Kanehisa. LIGAND: chemical database for enzyme reactions. *Bioinformatics (Oxford, England)*, 14(7):591–599, 1998.

[2] D. Lopez-Paz, P. Hennig, and B. Schölkopf. The randomized dependence coefficient. In *Advances in neural information processing systems*, pages 1–9, 2013.

[3] S. Filippi, C. C. Holmes, et al. A bayesian nonparametric approach to testing for dependence between random variables. *Bayesian Analysis*, 2017.

[4] T. Kunihama and D. B. Dunson. Nonparametric bayes inference on conditional independence. *Biometrika*, 103(1):35–47, 2016.

[5] F. Saad and V. Mansinghka. Detecting dependencies in sparse, multivariate databases using probabilistic programming and non-parametric bayes. In *Artificial Intelligence and Statistics*, pages 632–641, 2017.

[6] V. Mansinghka, P. Shafto, E. Jonas, C. Petschulat, M. Gasner, and J. B. Tenenbaum. CrossCat: A fully bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *arXiv preprint arXiv:1512.01272*, 2015.

[7] V. Mansinghka, R. Tibbetts, J. Baxter, P. Shafto, and B. Eaves. BayesDB: A probabilistic programming system for querying the probable implications of data. *arXiv preprint arXiv:1512.05006*, 2015.

[8] F. Obermeyer, J. Glidden, and E. Jonas. Scaling nonparametric bayesian inference via subsample-annealing. In *Artificial Intelligence and Statistics*, pages 696–705, 2014.

[9] F. Saad, L. Casarsa, and V. Mansinghka. Probabilistic search for structured data via probabilistic programming and nonparametric bayes. *arXiv preprint arXiv:1704.01087*, 2017.

[10] H. Rosling, A. Rosling Ronnlund, and H. Rosling. Gapminder: unveiling the beauty of statistics for a fact based world view. *URL https://www. gapminder. org/data*, 2010.

[11] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore. Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed research international*, 2014, 2014.

[12] F. Saad and V. K. Mansinghka. A probabilistic programming approach to probabilistic data analysis. In *Advances in Neural Information Processing Systems*, pages 2011–2019, 2016.

[13] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[14] C. E. Rasmussen. The infinite gaussian mixture model. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 554–560, Cambridge, MA, USA, 1999. MIT Press.