Probabilistic Search for Structured Data via Probabilistic Programming and Nonparametric Bayes

Feras Saad, Leonardo Casarsa, and Vikash Mansinghka
Probabilistic Computing Project
Massachusetts Institute of Technology

Abstract

Databases are widespread, yet extracting relevant data can be difficult. Without substantial domain knowledge, multivariate search queries often return sparse or uninformative results. This paper introduces an approach for searching structured data based on probabilistic programming and nonparametric Bayes. Users specify queries in a probabilistic language that combines standard SQL database search operators with an information theoretic ranking function called *predictive relevance*. Predictive relevance can be calculated by a fast sparse matrix algorithm based on posterior samples from CrossCat, a nonparametric Bayesian model for high-dimensional, heterogeneously-typed data tables. The result is a flexible search technique that applies to a broad class of information retrieval problems, which we integrate into BayesDB, a probabilistic programming platform for probabilistic data analysis. This paper demonstrates applications to databases of US colleges, global macroeconomic indicators of public health, and classic cars. We found that human evaluators often prefer the results from probabilistic search to results from a standard baseline.

1 Introduction

We are surrounded by multivariate data, yet it is difficult to search. Consider the problem of finding a university with a city campus, low student debt, high investment in student instruction, and tuition fees within a certain budget. The US College Scorecard dataset (Council of Economic Advisers, 2015) contains these variables plus hundreds of others. However, choosing thresholds for the quantitative variables — debt, investment, tuition, etc — requires domain knowledge. Furthermore, results grow sparse as more constraints are added. Figure 1a shows results from an SQL `SELECT` query with plausible thresholds for this question that yields only a single match.

This paper shows how to formulate a broad class of probabilistic search queries on structured data using probabilistic programming and information theory. The core technical idea combines SQL search operators with a ranking function called *predictive relevance* that assesses the relevance of database records to some set of query records, in a context defined by a variable of interest. Figures 1b and 1c show two examples, expanding and then refining the result from Figure 1a by combining predictive relevance with SQL. Predictive relevance is the probability that a candidate record is informative about the answers to a specific class of predictive queries about unknown fields in the query records.

The paper presents an efficient implementation applying a simple sparse matrix algorithm to the results of inference in CrossCat (Mansinghka et al., 2016). The result is a scalable, domain-general search technique for sparse, multivariate, structured data that combines the strengths of SQL search with probabilistic approaches to information retrieval. Users can query by example, using real records in the database if they are familiar with the domain, or partially-specified hypothetical records if they are less familiar. Users can then narrow search results by adding Boolean filters, and by including multiple records in the query set rather than a single record. An overview of the technique and its integration into BayesDB (Mansinghka et al., 2015) is shown in Figure 3.

We demonstrate the proposed technique with databases of (i) US colleges, (ii) public health and macroeconomic indicators, and (iii) cars from the late 1980s. The paper empirically confirms the scalability of the technique and shows that human evaluators often prefer results from the proposed technique to results from a standard baseline.

```
%bql SELECT
...   "institute",
...   "median_sat_math",
...   "admit_rate",
...   "tuition",
...   "median_student_debt",
...   "instructional_invest",
...   "locale"
... FROM college_scorecard
... WHERE
...   "locale" LIKE '%City%'
...   "tuition" < 50000
...   "median_student_debt" < 10000
...   "instructional_invest" > 50000
... LIMIT 10
```

(a) Standard SQL. Using a SQL `WHERE` clause to search for a university with a city campus, low student debt (at most $10K), high investment in student instruction (at least $50K), and a tuition within their budget (at most $50K). Due to sparsity in the dataset for the chosen thresholds, the Boolean conditions in the clause have only a single matching result, shown in the table below. The user needs to iteratively adjust the thresholds in order to obtain more results which match the search query.

institute	admit	sat	tuition	debt	investment	locale
Duke University	11%	745	47,243	7,500	50,756	Midsize City

```
%bql SELECT
...   "institute",
...   "admit_rate",
...   "median_sat_math",
...   "tuition",
...   "median_student_debt",
...   "instructional_invest",
...   "locale"
... FROM college_scorecard
... ORDER BY
...   RELEVANCE PROBABILITY
...   TO HYPOTHETICAL ROW ((
...    "locale" = 'Midsize City'
...    "tuition" = 50000,
...    "median_student_debt" = 10000,
...    "instructional_invest" = 50000
...   ))
...  IN THE CONTEXT OF
...   "instructional_invest"
... DESC
... LIMIT 10
```

(b) Relevance to hypothetical record. If the search query is instead specified as a hypothetical record in a BQL `RELEVANCE PROBABILITY` query, then `ORDER BY` can give the top-10 ranked matches. The results are all top-tier schools with high teaching investment, a city or large suburban campus, and low student debt. However, the user is surprised by the highly stringent admission rates at these colleges, which are mostly below 10%.

institute	admit	sat	tuition	debt	investment	locale
Duke University	11%	745	47,243	7,500	50,756	Midsize City
Princeton University	8%	755	41,820	7,500	52,224	Large Suburb
Harvard University	6%	755	43,938	6,500	49,500	Midsize City
Univ of Chicago	8%	758	49,380	12,500	83,779	Large City
Mass Inst Technology	8%	770	45,016	14,990	62,770	Midsize City
Calif Inst Technology	8%	785	43,362	11,812	92,590	Midsize City
Stanford University	5%	745	45,195	12,782	93,146	Large Suburb
Yale University	6%	750	45,800	13,774	107,982	Midsize City
Columbia University	7%	745	51,008	23,000	80,944	Large City
University of Penn.	10%	735	47,668	21,500	49,018	Large City

```
%bql SELECT
...   "institute",
...   "admit_rate",
...   "median_sat_math",
...   "tuition",
...   "median_student_debt",
...   "instructional_invest",
...   "locale"
... FROM college_scorecard
... WHERE
...   "admit_rate" > 0.10
...   AND "locale" LIKE '%City%'
... ORDER BY
...   RELEVANCE PROBABILITY
...   TO EXISTING ROWS IN (
...    'Duke University',
...    'Harvard University',
...    'Mass Inst Technology',
...    'Yale University',
...   )
...  IN THE CONTEXT OF
...   "instructional_invest"
... DESC
... LIMIT 10
```

(c) Relevance to observed records combined with SQL. Combining BQL and SQL to search for colleges which are most relevant to the schools from (b) in the context of "instructional investment", but that must have (i) less stringent admissions (at least 10%) and (ii) city campuses only. The quantitative search metrics of interest for the colleges in the result set are all significantly better than the national average, but they are mostly below the more selective schools in (b).

institute	admit	sat	tuition	debt	investment	locale
Duke University	11%	745	47,243	7,500	50,756	Midsize City
Georgetown Univ	17%	710	46,744	17,000	31,102	Midsize City
Johns Hopkins Univ	16%	730	47,060	16,250	77,339	Midsize City
Vanderbilt Univ	13%	760	43,838	13,000	79,372	Large City
University of Penn.	10%	735	47,668	21,500	49,018	Large City
Carnegie Mellon	24%	750	49,022	25,250	31,807	Midsize City
Rice University	15%	750	40,566	9,642	40,056	Midsize City
Univ Southern Calif	18%	710	48,280	21,500	43,170	Midsize City
Cooper Union	15%	710	41,400	18,250	21,635	Large City
New York University	35%	685	46,170	23,300	30,237	Large City

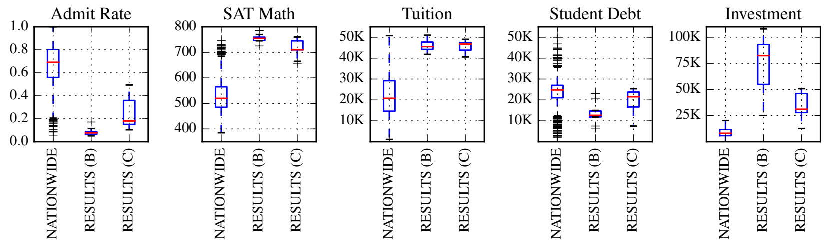


Figure 1: Combining predictive relevance probability in the Bayesian Query Language (BQL) with standard techniques in SQL to search the US College Scorecard dataset. The full data contains over 7000 colleges and 1700 variables, and is available for download at `collegescorecard.ed.gov/data`.

2 Establishing an information theoretic definition of context-specific predictive relevance

In this section, we outline the basic set-up and notations for the database search problem, and establish a formal definition of the probability of "predictive relevance" between records in the database.

2.1 Finding predictively relevant records

Suppose we are given a sparse dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ containing N records, where each $\mathbf{x}_r = (x_{[r,1]}, \dots, x_{[r,p]})$ is an instantiation of a p-dimensional random vector, possibly with missing values. For notational convenience, we refer to arbitrary collections of observations using sets as indices, so that $\mathbf{x}_{[R,C]} \equiv \{x_{[r,c]} : r \in R, c \in R\}$. Bold-face symbols denote multivariate entities, and variables are capitalized as $X_{[r,c]}$ when they are unobserved (i.e. random).

Let $\mathcal{Q} \subset [N]$ index a small collection of "query records" $\mathbf{x}_\mathcal{Q} = \{\mathbf{x}_q : q \in \mathcal{Q}\}$. Our objective is to rank each item $\mathbf{x}_i \in \mathcal{D}$ by how relevant it is for formulating predictions about values of $\mathbf{x}_\mathcal{Q}$, "in the context" of a particular dimension c. We formally define the context of c as a subset of dimensions $\mathcal{V} \subseteq [p]$ such that for an arbitrary record r^* and each $v \in \mathcal{V}$, the random variable $X_{[r^*,v]}$ is statistically dependent with $X_{[r^*,c]}$.[1]

In other words, we are searching for records i where knowledge of $\mathbf{x}_{[i,\mathcal{V}]}$ is useful for predicting $\mathbf{x}_{[\mathcal{Q},\mathcal{V}]}$, had we not known the values of these observations.

2.2 Defining context-specific predictive relevance using mutual information

We now formalize the intuition from the previous section more precisely. Let $\mathcal{R}_c(\mathcal{Q}, r)$ denote the probability that r is predictively relevant to $\mathcal{Q}$, in the context of c. Furthermore, let c^* denote the index of a new dimension in the length-p random vectors, which is statistically dependent on dimension c (i.e. is in its context) but is not one of the p existing variables in the database. Since c^* indexes a novel variable, its value for each row r is itself a random variable, which we denote $X_{[r,c^*]}$. We now define the probability that r is predictively relevant to $\mathcal{Q}$ in the context of c as the posterior probability that the mutual information of $X_{[r,c^*]}$ and each query record $X_{[q,c^*]}$ is non-zero:

$$\mathcal{R}_c(\mathcal{Q}, r) = \mathbb{P}\left[\bigcap_{q \in \mathcal{Q}} \left(\mathcal{I}(X_{[q,c^*]} : X_{[r,c^*]}) > 0 \right) \,\middle|\, \lambda_{c^*}, \alpha, \mathcal{D} \right]. \tag{1}$$

The symbol λ_{c^*} refers to an arbitrary set of hyperparameters which govern the distribution of dimension c^*, and α is a context-specific hyperparameter which controls the prior on structural dependencies between the random variables $\{X_{[r,c^*]} : r \in [N]\}$. Moreover, the mutual information $\mathcal{I}$, a well-established measure for the strength of predictive relationships between random variables (Cover and Thomas, 2012), is defined in the usual way,

$$\mathcal{I}(X_{[q,c^*]} : X_{[r,c^*]} \mid \lambda_{c^*}, \alpha, \mathcal{D}) = \mathbb{E}\left[\log \left(\frac{p(X_{[q,c^*]}, X_{[r,c^*]} | \lambda_{c^*}, \alpha, \mathcal{D})}{p(X_{[q,c^*]} | \lambda_{c^*}, \alpha, \mathcal{D}) p(X_{[r,c^*]} | \lambda_{c^*}, \alpha, \mathcal{D})} \right) \right]. \tag{2}$$

Figure 2 illustrates the predictive relevance probability in terms of a hypothesis test on two competing graphical models, where the mutual information is non-zero in panel (a) indicating predictive relevance; and zero in panel (b), indicating predictive irrelevance.

2.3 Related Work

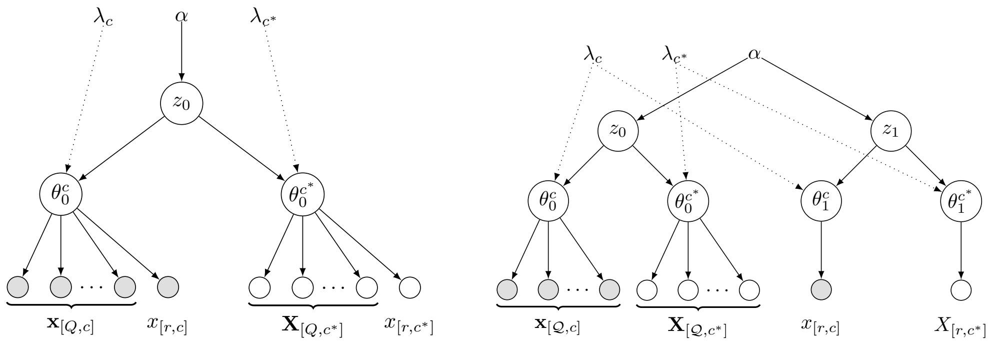
Our formulation of predictive relevance in terms of mutual information between new variables $X_{[r,c^*]}$ is related to the idea of "property induction" from the cognitive science literature (Rips, 1975; Osherson et al., 1990; Shafto et al., 2008), where subjects are asked to predict whether an entity has a property, given that some other entity has that property; e.g. how likely are cats to have some new disease, given that mice are known to have the disease?

It is also informative to consider the relationship between the predictive relevance $\mathcal{R}_c(\mathcal{Q}, r)$ in Eq (1) and the Bayesian Sets ranking function from the statistical modeling literature (Ghahramani and Heller, 2005):

$$\text{score}_{\text{Bayes-Sets}}(\mathcal{Q}, r) = \frac{p(\mathbf{x}_r | \mathbf{x}_\mathcal{Q})}{p(\mathbf{x}_r)}. \tag{3}$$

Bayes Sets defines a Bayes Factor, or ratio of marginal likelihoods, which is used for hypothesis testing without assuming a structure prior. On the other hand, predictive relevance defines a posterior probability, whose value is between 0 and 1, and therefore requires a prior over dependence structure between records (our approach outlined in Section 3 is based on nonparametric Bayes). While Bayes Sets draws inferences using only the query and candidate rows without considering the rest of the data, predictive relevance probabilities are necessarily

[1]A general definition for statistical dependence is having non-zero mutual information with the context variable. However, the method for detecting dependence to find variables in the context can be arbitrary e.g., using linear statistics such as Pearson-R, directly estimating mutual information, or others.



(a) Same generative process for $\mathbf{x}_Q$ and x_r. **(b)** Different generative processes for $\mathbf{x}_Q$ and x_r.

Figure 2: The predictive relevance of a collection of query records $\mathcal{Q}$ to a candidate record r, in the context of variable c, computes the probability that $\mathbf{x}_{[\mathcal{Q},c]}$ and $x_{[r,c]}$ are drawn from (a) the same generative process, versus (b) different generative processes. The latent variables z_0 and z_1 are indicators for the generative process of the records; and θ_0^c (resp. θ_1^c) are distributional parameters of data under model z_0 (resp. z_1) for variable c. Hyperparameter α dictates the prior on z, and λ dictates the prior on distributional parameters θ. The symbol c^* denotes a new dimension which is statistically dependent on c, and for which no values are observed for either $\mathcal{Q}$ or r. Conditioned on hyperparameters, knowing $X_{[r,c^*]}$ in (a) carries information about the unknown values $\mathbf{X}_{[\mathcal{Q},c^*]}$, whereas in (b) it does not.

conditioned on $\mathcal{D}$ as in Eq (1). Finally Bayes Sets considers the entire data vectors for scoring, whereas predictive relevance considers only dimensions which are in the context of a variable c, making it possible for two records to be predictively relevant in some context but probably predictively irrelevant in another.

3 Computing the probability of predictive relevance using nonparametric Bayes

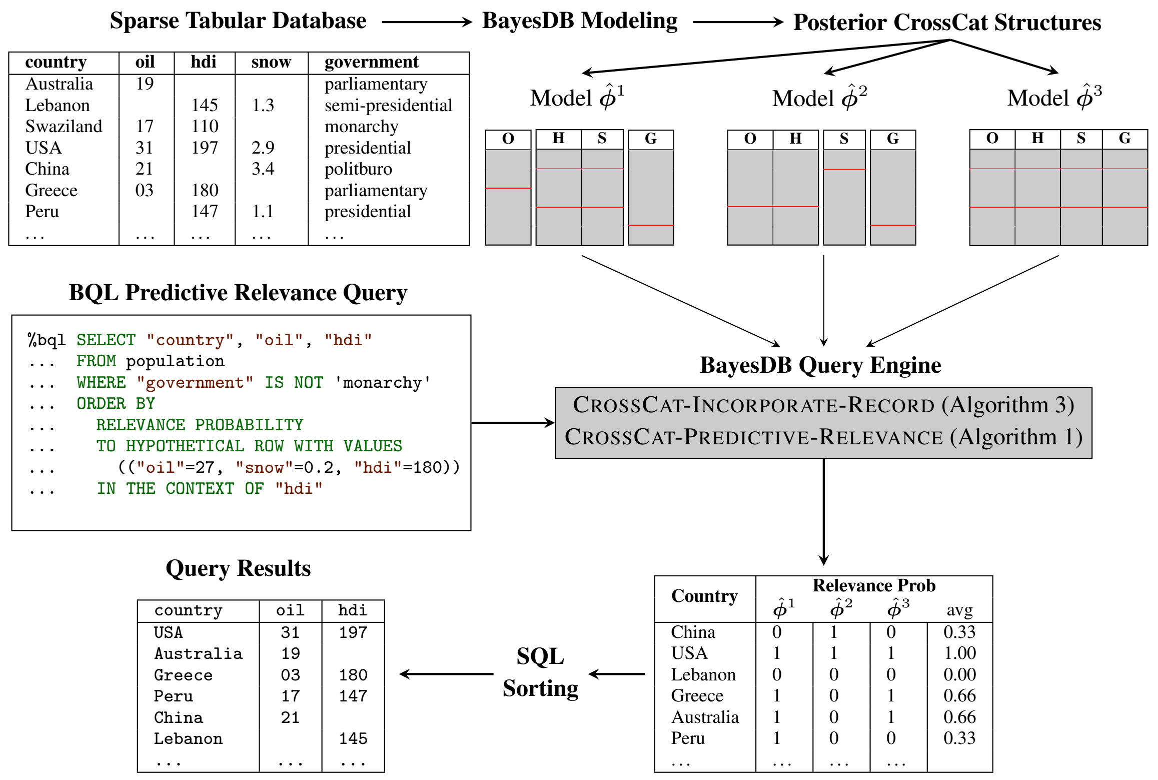
This section describes the cross-categorization prior (CrossCat, Mansinghka et al. (2016)) and outlines algorithms which use CrossCat to efficiently estimate predictive relevance probabilities Eq (1) for sparse, high-dimensional, and heterogenously-typed data tables.

CrossCat is a nonparametric Bayesian model which learns the full joint distribution of p variables using structure learning and divide-and-conquer. The generative model begins by partitioning the set of p variables into blocks using a Chinese restaurant process. This step is CrossCat's "outer" clustering, since it partitions the columns of a data table where variables correspond to columns, and records correspond to rows. Let π denote the partition of $[p]$ whose k-th block is $\mathcal{V}^k \subseteq [p]$: for $j \neq k$, all variables in $\mathcal{V}^k$ are mutually (marginally and conditionally) independent of all variables in $\mathcal{V}^j$. Within block k, the variables $\mathbf{x}_{[r,\mathcal{V}^k]}$ follow a Dirichlet process mixture model (Escobar and West, 1995), where we focus on the case the joint distribution factorizes given the latent cluster assignment z_r^k. This step is an "inner" clustering in CrossCat, since it specifies a cluster assignment for each row in block k. CrossCat's combinatorial structure requires detailed notation to track the latent variables and dependencies between them. The generative process for an exchangeable sequence $(\mathbf{X}_1, \dots, \mathbf{X}_N)$ of N random vectors is summarized below.

Table 1: Symbols used to describe CrossCat prior

Symbol	Description
α_0	Concentration hyperparameter of column CRP
α_1	Concentration hyperparameter of row CRP
v_c	Index of variable c in column partition
$\mathcal{V}^k$	List of variables in block k of column partition
z_r^k	Cluster index of r in row partition of block k
$\mathcal{C}_y^k$	List of rows in cluster y of block k
M_c	Joint distribution of data for variable c
λ_c	Hyperparameters of M_c
$X_{[r,c]}$	r-th observation of variable c
$\textsc{Set}(l)$	Unique items in list l

```
CrossCat Prior
1. Sample column partition into blocks.
   $\mathbf{v} = (v_1, \dots, v_p) \sim \textsc{Crp}(\cdot|\alpha_0)$
   $\mathcal{V}^k \leftarrow \{c \in [p] : v_c = k\}$          foreach $k \in \textsc{Set}(\mathbf{v})$
2. Sample row partitions within each block.
   $\mathbf{z}^k = (z_1^k, \dots, z_N^k) \sim \textsc{Crp}(\cdot|\alpha_1)$   foreach $k \in \textsc{Set}(\mathbf{v})$
   $\mathcal{C}_y^k \leftarrow \{r \in [N] : z_r^k = y\}$          foreach $k \in \textsc{Set}(\mathbf{v})$
                                                foreach $y \in \textsc{Set}(\mathbf{z}^k)$
3. Sample data jointly within row cluster.
   $\{X_{[r,c]} : r \in \mathcal{C}_y^k\} \sim M_c(\cdot|\lambda_c)$      foreach $k \in \textsc{Set}(\mathbf{v})$
                                                foreach $y \in \textsc{Set}(\mathbf{z}^k)$
                                                  foreach $c \in \mathcal{V}^k$
```



country	oil	hdi	snow	government
Australia	19			parliamentary
Lebanon		145	1.3	semi-presidential
Swaziland	17	110		monarchy
USA	31	197	2.9	presidential
China	21		3.4	politburo
Greece	03	180		parliamentary
Peru		147	1.1	presidential
...	...	...	...	...

```
%bql SELECT "country", "oil", "hdi"
...  FROM population
...  WHERE "government" IS NOT 'monarchy'
...  ORDER BY
...    RELEVANCE PROBABILITY
...    TO HYPOTHETICAL ROW WITH VALUES
...      (("oil"=27, "snow"=0.2, "hdi"=180))
...    IN THE CONTEXT OF "hdi"
```

Country	Relevance Prob			
	$\hat{\phi}^1$	$\hat{\phi}^2$	$\hat{\phi}^3$	avg
China	0	1	0	0.33
USA	1	1	1	1.00
Lebanon	0	0	0	0.00
Greece	1	0	1	0.66
Australia	1	0	1	0.66
Peru	1	0	0	0.33
...	...	...	...	

country	oil	hdi
USA	31	197
Australia	19	
Greece	03	180
Peru	17	147
China	21	
Lebanon		145
...	...	...

Figure 3: BayesDB workflow for computing context-specific predictive relevance between database records. Modeling and inference in BayesDB produces an ensemble of posterior CrossCat model structures. Each structure specifies (i) a column partition for the factorization of the joint distribution of all variables in the database, using a Chinese restaraunt process; and (ii) a separate row partition within each block of variables, using a Dirichlet process mixture. The column partition clusters variables into different "contexts", where all variables in a context are probably dependent on one another. With each context, the row partition clusters records which are probably informative of one another. End-user queries for predictive relevance are expressed in Bayesian Query Langauge. The BQL interpreter aggregates relevance probabilities across the ensemble, and can use them as a ranking function in a probabilistic `ORDER BY` query.

The representation of CrossCat in this paper assumes that data within a cluster is sampled jointly (step 3), marginalizing over cluster-specific distributional parameters:

$$M_c(\mathbf{x}_{[\mathcal{C}^k_y,c]}, \lambda_c) = \int_\theta \prod_{r \in \mathcal{C}^k_y} p(x_{[r,c]}|\theta)p(\theta|\lambda_c)d\theta.$$

This assumption suffices for our development of predictive relevance, and is applicable to a broad class of statistical data types (Saad and Mansinghka, 2016) with conjugate prior-likelihood representations such as Beta-Bernoulli for binary, Dirichlet-Multinomial for categorical, Normal-Inverse-Gamma-Normal for real values, and Gamma-Poisson for counts.

Given dataset $\mathcal{D}$, we refer to Obermeyer et al. (2014) and Mansinghka et al. (2016) for scalable algorithms for posterior inference in CrossCat, and assume we have access to an ensemble of H posterior samples $\left\{\hat{\phi}^1, \dots, \hat{\phi}^H\right\}$ where each $\hat{\phi}^h$ is a realization of all variables in Table 1.

3.1 Estimating predictive relevance using CrossCat

We now describe how to use posterior samples of CrossCat to efficiently estimate the predictive relevance probability $\mathcal{R}_c(\mathcal{Q}, r)$ from Eq (1). Letting c denote the context variable, we formalize the novel variable c^* as a fresh column in the tabular population which is assigned to the same block k as c (i.e. $k = v_c = v_{c^*}$). As shown by Saad and Mansinghka (2017), structural dependencies induced by CrossCat's variable partition are related to an upper-bound on the probability there exists a statistical dependence between c and c^*. To estimate Eq (1), we first treat the mutual information between $X_{[q,c^*]}$ and $X_{[r,c^*]}$ as a derived random variable, which is a function of their random cluster assignments z^k_q and z^k_r,

$$(z^k_q, z^k_r) \mapsto \mathcal{I}(X_{[q,c^*]} : X_{[r,c^*]}|z^k_q, z^k_r, \alpha_1, \lambda_{c^*}). \quad (4)$$

The key insight, implied by step 3 of the CrossCat prior, is that, conditioned on their assignments, rows from different clusters are sampled independently, which gives

$$\begin{aligned} & z_q^k \neq z_r^k \\ & \iff p(x_{[q,c^*]}, x_{[r,c^*]} | z_q^k, z_r^k, \lambda_{c^*}, \alpha_1, \mathcal{D}) = \\ & \qquad p(x_{[q,c^*]} | z_q^k, \lambda_{c^*}, \alpha_1, \mathcal{D}) p(x_{[r,c^*]} | z_r^k, \lambda_{c^*}, \alpha_1, \mathcal{D}) \\ & \iff \mathcal{I}(X_{[q,c^*]} : X_{[r,c^*]} | z_q^k, z_r^k, \alpha_1, \lambda_{c^*}) = 0, \end{aligned} \tag{5}$$

where the final implication follows directly from the definition of mutual information in Eq (2). Note that Eq (5) does not depend on the particular choice of λ_{c^*}, and indeed this hyperparameter is never represented explicitly. Moreover, hyperparameter α_1 (corresponding to α in Figure 2) is the concentration of the Dirichlet process for CrossCat row partitions.

Eq (5) implies that we can estimate the probability of non- zero mutual information between $X_{[r,c^*]}$ and each $X_{[q,c^*]}$ for $q \in \mathcal{Q}$ by forming a Monte Carlo estimate from the ensemble of posterior CrossCat samples,

$$\begin{aligned} & \mathcal{R}_c(\mathcal{Q}, r) \\ & = \mathbb{P}\left[\bigcap_{q \in \mathcal{Q}} \left(\mathcal{I}(X_{[q,c^*]} : X_{[r,c^*]}) > 0 \right) \middle| \lambda_{c^*}, \alpha_1, \mathcal{D} \right] \\ & = \mathbb{P}\left[\bigcap_{q \in \mathcal{Q}} \left(z_q^{v_c} = z_r^{v_c} \right) \middle| \alpha_1, \mathcal{D} \right] \\ & \approx \frac{1}{H} \sum_{h=1}^{H} \left[\mathbb{I}\left[\bigcap_{q \in \mathcal{Q}} \left(\hat{z}_q^{\hat{v}_c^h, h} = \hat{z}_r^{\hat{v}_c^h, h} \right) \right] \right], \end{aligned} \tag{6}$$

where $\hat{v}_c^h$ indexes the context block, and $\hat{z}_r^{\hat{v}_c^h,h}$ denotes cluster assignment of r in the row partition of $\hat{v}_c^h$, according to the sample $\hat{\phi}^h$. Algorithm 1 outlines a procedure (used by the BayesDB query engine from Figure 3) for formulating a Monte Carlo based estimator for a predictive relevance query using CrossCat.

Algorithm 1 CROSSCAT-PREDICTIVE-RELEVANCE

```
Require: { CrossCat samples: $\hat{\phi}^h$ for $h = 1,\ldots,H$
         { query rows: $\mathcal{Q} = \{q_i : 1 \le i \le |\mathcal{Q}|\}$
         { context variable: $c$
Ensure: predictive relevance of each existing row in $\mathcal{D}$ to $\mathcal{Q}$
 1: for $r = 1,\ldots,N$ do                        ▷ for each existing row
 2:   for $h = 1,\ldots,H$ do                      ▷ for each CrossCat sample
 3:     $k \leftarrow \hat{v}_c^h$                  ▷ retrieve the context block
 4:     for $q \in \mathcal{Q}$ do                  ▷ for each query row
 5:       if $\hat{z}_q^{k,h} \neq \hat{z}_r^{k,h}$ then   ▷ $r$ and $q$ are different clusters
 6:         $\mathcal{R}_c^h(\mathcal{Q}, r) \leftarrow 0$   ▷ $r$ irrelevant to some $q$
 7:         break
 8:     else                                        ▷ $r$ in same cluster as all $q \in \mathcal{Q}$
 9:       $\mathcal{R}_c^h(\mathcal{Q}, r) \leftarrow 1$     ▷ $r$ relevant to all $q$
10:   $\mathcal{R}_c(\mathcal{Q}, r) \leftarrow \frac{1}{H}\sum_{h=1}^{H} \mathcal{R}_c^h(\mathcal{Q}, r)$   ▷ average relevances
11: return $\{\mathcal{R}_c(\mathcal{Q}, r) : 1 \le r \le N\}$
```

3.2 Optimizing the estimator using a sparse matrix-vector multiplication

In this section, we show how to greatly optimize the naive, nested for-loop implementation in Algorithm 1 by instead computing predictive relevance for all r through a single matrix-vector multiplication.

Define the pairwise cluster co-occurrence matrix $\mathbf{S}^{k,h}$ for block k of CrossCat sample $\hat{\phi}^h$ to have binary entries $\mathbf{S}_{i,j}^{k,h} = \mathbb{I}[\hat{z}_i^{k,h} = \hat{z}_j^{k,h}]$. Furthermore, let $\mathbf{1}_{\mathcal{Q}}$ denote a length-N vector with a 1 at indexes $q \in \mathcal{Q}$ and 0 otherwise. We vectorize $\mathcal{R}_c(\mathcal{Q}, r)$ across $r \in [N]$ by:

$$\mathbf{u}^h = \frac{1}{|\mathcal{Q}|} \mathbf{S}^{k,h} \mathbf{1}_{\mathcal{Q}} \qquad h = 1, \ldots, H \tag{7}$$

$$\mathcal{R}_c(\mathcal{Q}, \cdot) = \frac{1}{H} \sum_{h=1}^{H} \mathbf{u}^h. \tag{8}$$

The resulting length-N vector $\mathbf{u}^h$ in Eq (7) satisfies $\mathbf{u}_r^h = 1$ if and only if $\hat{z}_r^{k,h} = \hat{z}_q^{k,h}$ for all $q \in \mathcal{Q}$, which we identify as the argument of the indicator function in Eq (6). Finally, by averaging $\mathbf{u}^h$ across the H samples in Eq (8), we arrive at the vector of relevance probabilities.

For large datasets, constructing the $N \times N$ matrix $\mathbf{S}^{k,h}$ using $\Theta(N^2)$ operations is prohibitively expensive. Algorithm 2 describes an efficient procedure that exploits CrossCat's sparsity to build $\mathbf{S}^{k,h}$ in expected time $\ll O(N^2)$ by using (i) a sparse matrix representation, and (ii) CrossCat's partition data structures to avoid considering all pairs of rows. This fast construction means that Eq (7) is practical to implement for large data tables.

The algorithm's running time depends on (i) the number of clusters $|\textsc{Set}(\hat{\mathbf{z}}^k)|$ in line 1; (ii) the average number of rows per cluster $|\hat{\mathcal{C}}_y^k|$ in line 2; and (iii) the data structures used to represent $\mathbf{S}^{k,h}$ in line 3. Under the CRP prior, the expected number of clusters is $O(\alpha_1 \log(N))$, which implies an average occupancy of $O(N/(\alpha_1 \log(N)))$ rows per cluster. If the sparse binary matrix is stored with a list-of-lists representation, then the update in line 3 requires $O(1)$ time. Furthermore, we emphasize that since $\mathbf{S}^{k,h}$ does not depend $\mathcal{Q}$, its cost of construction is amortized over an arbitrary number of queries.

Algorithm 2 CROSSCAT-CO-OCCURRENCE-MATRIX

```
Require: CrossCat sample $\hat{\phi}^h$; block index $k$.
Ensure: Pairwise co-occurrence matrix $\mathbf{S}^{k,h}$
 1: for $y \in \textsc{Set}(\hat{\mathbf{z}}^k)$ do        ▷ for each cluster in block $k$
 2:   for $r \in \hat{\mathcal{C}}_y^k$ do                 ▷ for each row in the cluster
 3:     Set $\mathbf{S}_{r,j}^{k,h} = 1$, where $j \in \hat{\mathcal{C}}_y^k$   ▷ update the matrix
 4: return $\mathbf{S}^{k,h}$
```

3.3 Computing predictive relevance probabilities for query records that are not in the database

We have so far assumed that the query records must consist of items that already exist in the database. This section relaxes this restrictive assumption by illustrating how to compute relevance probabilities for search records which do not exist in $\mathcal{D}$, and are instead specified by the user on a per-query basis (refer to the BQL query in Figure 3 for an example of a hypothetical query record). The key idea is to (i) incorporate the new records into each CrossCat sample $\hat{\phi}^h$ by using a Gibbs-step to sample cluster assignments from the joint posterior (Neal, 2000); (ii) compute Eq (7) on the updated samples; and (iii) unincorporate the records, leaving the original samples unmutated.

Letting $\{\mathbf{x}_{[N+i]} : 1 \le i \le t\}$ denote t (partially observed) new rows and $\mathcal{Q} = \{N+1, \dots, N+t\}$ the query, we compute $\mathcal{R}_c(\mathcal{Q}, r)$ for all r by first applying CrossCat-Incorporate-Record (Algorithm 3) to each $q \in \mathcal{Q}$ sequentially. Sequential incorporation corresponds to sampling from the sequence of predictive distributions, which, by exchangeability, ensures that each updated $\hat{\phi}^h$ contains a sample of cluster assignments from the joint distribution, guaranteeing correctness of the Monte Carlo estimator in Eq (6). Note that since CrossCat specifies a non-parametric mixture, the proposal clusters include all existing clusters, plus one singleton cluster $\max(\mathbf{z}^k)+1$. We next update the co-occurrence matrices in time linear in the size of the sampled cluster and then evaluate Eq (7) and (8). To unincorporate, we reverse lines 7-9 and restore the co-occurrence matrices. Figure 4 confirms that the runtime scaling is asymptotically linear, varying the (i) number of new rows, (ii) fraction of variables specified for the new rows that are in the context block (i.e. query sparsity), (iii) number of clusters in the context block, and (iv) number of variables in the context block.

Algorithm 3 CrossCat-Incorporate-Record

```
Require: CrossCat sample $\phi$; context $c$; new row $\mathbf{x}_{N+1}$
Ensure: Updated crosscat sample $\phi'$
 1: $k \leftarrow v_c$                                   ▷ Retrieve block of context variable
 2: $Y \leftarrow \max(\mathbf{z}^k) + 1$                ▷ Retrieve proposal clusters
 3: for $y = 1, \dots, Y$ do                             ▷ Compute cluster probabilities
 4:     $n_y \leftarrow \begin{cases} |\mathcal{C}^k_y| & \text{if } y \in \mathbf{z}^k \\ \alpha_1 & \text{if } y = \max(\mathbf{z}^k)+1 \end{cases}$
 5:     $l_y \leftarrow \left(\prod_{c \in \mathcal{V}^k} M_c(x_{[N+1,c]} | \mathbf{x}_{[\mathcal{C}^k_y, c]}, \lambda_c)\right) n_y$
 6: $z^k_{N+1} \sim \text{Categorical}(l_1, \dots, l_Y)$          ▷ Sample cluster
 7: $\mathbf{z}'^k \leftarrow \mathbf{z}^k \cup \{z^k_{N+1}\}$     ▷ Append cluster assignment
 8: $\mathcal{C}'^k_{z^k_{N+1}} \leftarrow \mathcal{C}^k_{z^k_{N+1}} \cup \{N+1\}$   ▷ Append row to cluster
 9: $\mathcal{D}' \leftarrow \mathcal{D} \cup \{\mathbf{x}_{[N+1, \mathcal{V}^k]}\}$   ▷ Append record to database
10: return $\phi'$                                       ▷ Return the updated sample
```

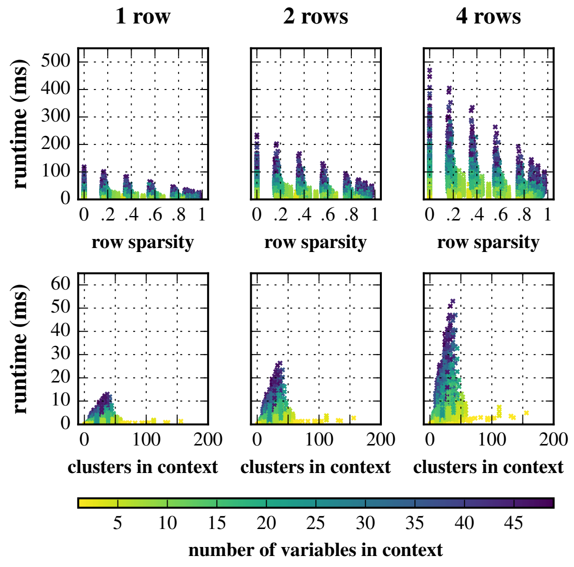


Figure 4: Empirical measurements of the asymptotic scaling of CrossCat-Incorporate-Record (Algorithm 3) on the Gapminder dataset (Section 4). The color of each measurement indicates the number of variables in the block of the context variable; each column shows a different number of records (1, 2, 4, and 8) incorporated by the algorithm. The top panels shows that, for a fixed number of variables in the context, the runtime (in milliseconds) decays linearly with the sparsity of the hypothetical records (dimensions which are not in the same block as the context variable are ignored). The lower panels show the runtime increasing linearly with the number of clusters in the context; the number of variables in the context dictates the slope of the curve.

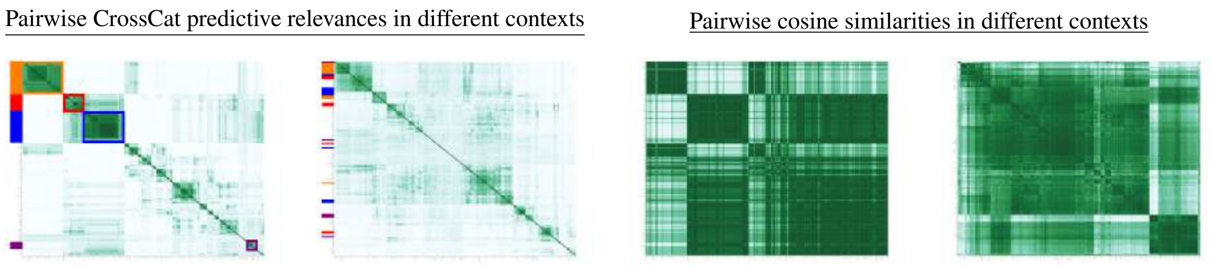
4 Applications

This section illustrates the efficacy of predictive relevance in BayesDB by applying the technique to several search problems in real-world, sparse, and high-dimensional datasets of public interest.[2]

4.1 College Scorecard

The College Scorecard (Council of Economic Advisers, 2015) is a federal dataset consisting of over 7000 colleges and 1700 variables, and is used to measure and improve the performance of US institutions of higher education. These variables include a broad set of categories

[2]Appendix D contains a further application to a dataset of classic cars from 1987. Appendix A formally describes the integration of `RELVANCE PROBABILITY` into BayesDB as an expression in the Bayesian Query Language (Figure 3).



(a) CrossCat (life expectancy) **(b)** CrossCat (exports, % gdp) **(c)** Cosine (life expectancy) **(d)** Cosine (exports, % gdp)

Concept	**Representative Countries in the Concept**
Low-Income Nations	Burundi, Ethiopia, Uganda, Benin, Malawi, Rwanda, Togo, Guinea, Senegal, Afghanistan, Malawi
Post-Soviet Nations	Russia, Ukraine, Bulgaria, Belarus, Slovakia, Serbia, Croatia, Poland, Hungary, Romania, Latvia
Western Democracies	France, Britain, Germany, Netherlands, Italy, Denmark, Finland, Sweden, Norway, Australia, Japan
Small Wealthy Nations	Qatar, Bahrain, Kuwait, Emirates, Singapore, Israel, Gibraltar, Bermuda, Jersey, Cayman Islands

(e) Countries which are mutually predictive in the context of "life expectancy" according to CrossCat's relevance matrix (a).

Figure 5: (a) – (d) Pairwise heatmaps of countries from the Gapminder dataset in the contexts of "life expectancy at birth" and "exports of goods and services (% of gdp) ", using CrossCat predictive relevance and cosine similarity. Each row and column in a matrix is a country, and a cell value (between 0 and 1) indicates the strength of match between those two countries. **(e)** CrossCat learns a sparse set of relevances; for "life expectancy", these broadly correspond to common-sense taxonomies of countries based on shared geographic, political and macroeconomic characteristics. These concepts were manually labeled by inspecting clusters of countries in matrix (a); the colors in the matrix correspond to countries in the table which belong to the concept of that color. Note that the relevance structure differs significantly when ranking in the context of "exports, % gdp", as shown by the colors in matrix (b) where the clusters of mutually relevant countries form a different pattern than in (a). Cosine similarity learns dense, noisy sets of spuriously high-ranking countries with coarser structure, as shown in (c) and (d). Refer to Appendix C for more baselines.

such as the campus characteristics, academic programs, student debt, tuition fees, admission rates, instructional investments, ethnic distributions, and completion rates. We analyzed a subset of 2000 schools (four-year institutions) and 100 variables from the categories listed above.

Suppose a student is interested in attending a city university with a set of desired specifications. Starting with a standard SQL Boolean search in Figure 1a (on p. 2) they find only one matching record, which requires iteratively rewriting the search conditions to retrieve more results.

Figure 1b instead expresses the search query as a hypothetical row in a BQL `PREDICTIVE RELEVANCE` query (which invokes the technique in Section 3.3). The top-ranking records contain first-rate schools, but their admission rates are much too stringent. In Figure 1c, the user re-expresses the BQL query to rank schools by predictive relevance, in the context of instructional investment, to a subset of the first-rate schools discovered in 1b. Combining `ORDER BY PREDICTIVE RELEVANCE` with Boolean conditions in the `WHERE` clause returns another set of top-quality schools with city-campuses that are less competitive than those in 1b, but have quantitative metrics that are much better than national averages.

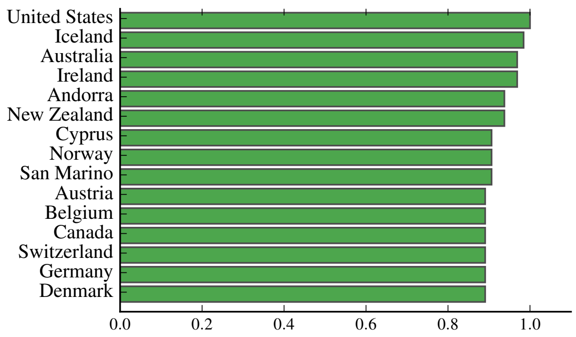
4.2 Gapminder

Gapminder (Rosling, 2008) is an extensive longitudinal dataset of over ∼320 global macroeconomic variables of population growth, education, climate, trade, welfare and health for 225 countries. Our experiments are based on a cross-section of the data from the year 2002. The data is sparse, with 35% of the data missing. Figure 5 shows heatmaps of the pairwise predictive relevances for all countries in the dataset under different contexts, and compares the results to cosine similarity. Clusters of predictively relevant countries form common-sense taxonomies; refer to the caption for further discussion.

Figure 6 finds the top-15 countries in the dataset ordered by their predictive relevance to the United States, in the context of "life expectancy at birth". Table 6b shows representative variables which are in the context; these variables have the highest dependence probability with the context variable, according a Monte Carlo estimate using 64 posterior CrossCat samples. The countries in Figure 6a are all rich, Western democracies with highly developed economies and advanced healthcare systems.

To quantitatively evaluate the quality of top-ranked countries returned by predictive relevance, we ran the tech-

```
%bql .barplot
...   ESTIMATE "country",
...     RELEVANCE PROBABILITY
...       TO EXISTING ROWS IN
...       ('United States')
...       IN THE CONTEXT OF
...       "life expectancy at birth"
...     AS "rel_us_lifexp"
...   FROM gapminder
...   ORDER BY "rel_us_lifexp" DESC
...   LIMIT 15
```



(a) Relevance to USA in the context of "life expectancy"

Measles, mumps, & rubella vaccines (% population)
Under 5 mortality rate
Dead children per woman
access to improved sanitation facilities (% population)
access to improved drinking water sources (% population)
human development index
body mass index (kg/m2)
murder rate (per 100,000)
food supply (kilocalories per person)
contraceptive prevalence (% women ages 15-49)
alcohol consumption (liters per adult)
prevalence of tobacco use among adults (% population)

(b) Variables in the context of "life expectancy at birth"

Figure 6: Using BQL to search for the top 15 countries in the Gapminder dataset ranked by their relevance to the United States in the context of "life expectancy at birth" finds rich, Western democracies with advanced healthcare systems.

nique on 10 representative search queries (varying the country and context variable) and obtained the top 10 results for each query. Figure 7 shows the queries, and human preferences for the results from predictive relevance versus results from cosine similarity between the country vectors. We defined the context for cosine similarity by the 320-dimensional vectors down to 10 dimensions and selecting variables which are most dependent with the context variable according to CrossCat's dependence probabilities. To deal with sparsity, which cosine similarity cannot handle natively, we imputed missing values using sample medians; imputation techniques like MICE (Buuren and Groothuis-Oudshoorn, 2011) resulted in little difference (Appendix C).

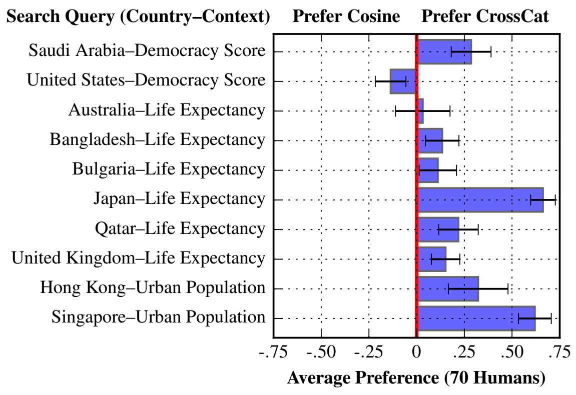


Figure 7: Comparing human preferences for the top-ranked countries returned by cosine similarity versus CrossCat predictive relevance, in 10 representative search queries (shown on the y-axis). For each query, human subjects were given the top 10 most relevant countries, according to both cosine and CrossCat, and then asked to choose which results they preferred, if any. We scored the responses in the following way: "countries returned by cosine are more relevant" (score = -1); "countries returned by CrossCat are more relevant" (score = +1); "both results are equally relevant" (score = 0). The x-axis shows the scores averaged across 70 humans, surveyed on the cloud through `crowdflower.com`. Error bars represent one standard error of the mean. For most of the queries, human preferences are biased in favor of CrossCat's rankings. Further details on the experimental design and results are given in Appendix B.

5 Discussion

This paper has shown how to perform probabilistic searches of structured data by combining ideas from probabilistic programming, information theory, and non-parametric Bayes. The demonstrations suggest the technique can be effective on sparse, real-world databases from multiple domains and produce results that human evaluators often preferred to a standard baseline.

More empirical evaluation is clearly needed, ideally including tests of hundreds or thousands of queries, more complex query types, and comparisons with query results manually provided by human domain experts. In fact, search via predictive relevance in the context of variables drawn from learned representations of data could potentially provide a meaningful way to compare representation learning techniques. It also may be fruitful to build a distributed implementation suitable for database representations of web-scale data, including photos, social network users, and web pages.

Relatively unstructured probabilistic models, such as topic models, proved sufficient for making unstructured text data far more accessible and useful. We hope this paper helps illustrate the potential for structured probabilistic models to improve the accessibility and usefulness of structured data.

Acknowledgments

The authors wish to acknowledge Ryan Rifkin, Anna Comerford, Marie Huber, and Richard Tibbetts for helpful comments on early drafts. This research was supported by DARPA (PPAML program, contract number FA8750-14-2-0004), IARPA (under research contract 2015-15061000003), the Office of Naval Research (under research contract N000141310333), the Army Research Office (under agreement number W911NF-13-1-0212), and gifts from Analog Devices and Google.

References

Stef Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3), 2011.

Council of Economic Advisers. Using federal data to measure and improve the performance of u.s. institutions of higher education. Technical report, Executive Office of the President of the United States, 2015.

Thomas Cover and Joy Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley, 2012.

Michael Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995.

Zoubin Ghahramani and Katherine A. Heller. Bayesian sets. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, pages 435–442. MIT Press, 2005.

Dennis Kibler, David W Aha, and Marc K Albert. Instance-based prediction of real-valued attributes. *Computational Intelligence*, 5(2):51–57, 1989.

Vikash Mansinghka, Richard Tibbetts, Jay Baxter, Pat Shafto, and Baxter Eaves. BayesDB: A probabilistic programming system for querying the probable implications of data. *CoRR*, abs/1512.05006, 2015.

Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua B. Tenenbaum. CrossCat: A fully Bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *Journal of Machine Learning Research*, 17(138):1–49, 2016.

Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

Fritz Obermeyer, Jonathan Glidden, and Eric Jonas. Scaling nonparametric Bayesian inference via subsample-annealing. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 696–705. JMLR.org, 2014.

Daniel N Osherson, Edward E Smith, Ormond Wilkie, Alejandro Lopez, and Eldar Shafir. Category-based induction. *Psychological review*, 97(2):185, 1990.

Lance J. Rips. Inductive judgments about natural categories. *Journal of Verbal Learning and Verbal Behavior*, 14(6):665–681, 1975.

Hans Rosling. Gapminder: Unveiling the beauty of statistics for a fact based world view, 2008. URL `https://www.gapminder.org/data/`.

Feras Saad and Vikash Mansinghka. Probabilistic data analysis with probabilistic programming. *CoRR*, abs/1608.05347, 2016.

Feras Saad and Vikash Mansinghka. Detecting dependencies in sparse, multivariate databases using probabilistic programming and non-parametric bayes. In *Proceedings of the Twentieth International Conference on Artificial Intelligence and Statistics*. JMLR.org, 2017.

Patrick Shafto, Charles Kemp, Elizabeth Bonawitz, John Coley, and Joshua Tenenbaum. Inductive reasoning about causally transmitted properties. *Cognition*, 109 (2):175–192, 2008.

Appendices

A Integrating predictive relevance as a ranking function in BayesDB

This section describes the integration of predictive relevance into BayesDB (Mansinghka et al., 2015; Saad and Mansinghka, 2016), a probabilistic programming platform for probabilistic data analysis.

New syntaxes in the Bayesian Query Language (BQL) allow a user to express predictive relevance queries where the query set can be an arbitrary combination of existing and hypothetical records. We implement predictive relevance in BQL as an expression with the following syntaxes, depending on the specification of the query records.

- Query records are existing rows.

```
RELEVANCE PROBABILITY
  TO EXISTING ROWS IN <expression>
  IN THE CONTEXT OF <context-var>
```

- Query records are hypothetical rows.

```
RELEVANCE PROBABILITY
  TO HYPOTHETICAL ROWS WITH VALUES (<values>)
  IN THE CONTEXT OF <context-var>
```

- Query records are existing and hypothetical rows.

```
RELEVANCE PROBABILITY
  TO EXISTING ROWS IN <expression>
  AND HYPOTHETICAL ROWS WITH VALUES (<values>)
  IN THE CONTEXT OF <context-var>
```

The expression is formally implemented as a 1-row BQL estimand, which specifies a map $r \mapsto \mathcal{R}_c(\mathcal{Q}, r)$ for each record in the table. As shown in the expressions above, query records are specified by the user in two ways: (i) by giving a collection of `EXISTING ROWS`, whose primary key indexes are either specified manually, or retrieved using an arbitrary BQL `<expression>`; (ii) by specifying one or more `HYPOTHETICAL RECORDS` with their `<values>` as a list of column-value pairs. These new rows are first incorporated using Algorithm 3 from Section 3.3 and they are then unincorporated after the query is finished. The `<context-var>` can be any variable in the tabular population.

As a 1-row function in the structured query language, the `RELEVANCE PROBABILITY` expression can be used in a variety of settings. Some typical use-cases are shown in the following examples, where we use only existing query rows for simplicity.

- As a column in an `ESTIMATE` query.

```
ESTIMATE
  "rowid",
  RELEVANCE PROBABILITY
    TO EXISTING ROWS IN <expression>
    IN THE CONTEXT OF <context-var>
  FROM <table>
```

- As a filter in `WHERE` clause.

```
ESTIMATE
  "rowid"
FROM <table>
WHERE (
    RELEVANCE PROBABILITY
      TO EXISTING ROWS IN <expression>
      IN THE CONTEXT OF <context-var>
    ) > 0.5
```

- As a comparator in an `ORDER BY` clause.

```
ESTIMATE
  "rowid"
FROM <table>
ORDER BY
    RELEVANCE PROBABILITY
      TO EXISTING ROWS IN <expression>
      IN THE CONTEXT OF <context-var>
[ASC | DESC]
```

It is also possible to perform arithmetic operations and Boolean comparisons on relevance probabilities.

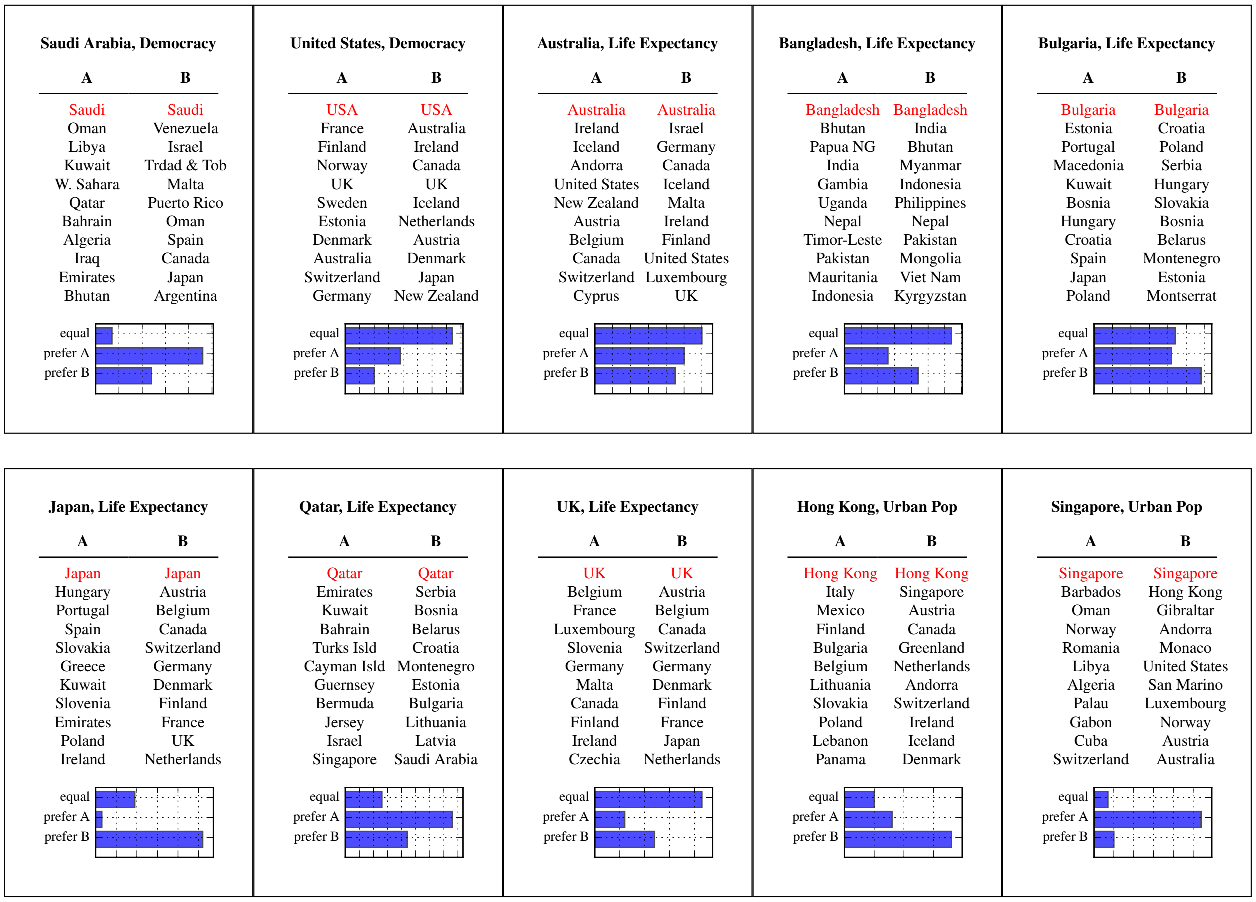
- Finding the mean relevance probability for a set of `rowid`s of interest.

```
ESTIMATE
  AVG (
    RELEVANCE PROBABILITY
      TO EXISTING ROWS IN <expression>
      IN THE CONTEXT OF <context-var>
  )
FROM <table>
WHERE "rowid" IN <expression>
```

- Finding rows which are more relevant in some context c_0 than in another context c_1.

```
ESTIMATE
  "rowid"
FROM <table>
WHERE (
    RELEVANCE PROBABILITY
      TO EXISTING ROWS IN <expression>
      IN THE CONTEXT OF <context-var-0>
    ) > (
    RELEVANCE PROBABILITY
      TO EXISTING ROWS IN <expression>
      IN THE CONTEXT OF <context-var-1>
    )
```

B Predictive relevance and cosine similarity on Gapminder human evaluation queries



Saudi Arabia, Democracy

A	B
Saudi	Saudi
Oman	Venezuela
Libya	Israel
Kuwait	Trdad & Tob
W. Sahara	Malta
Qatar	Puerto Rico
Bahrain	Oman
Algeria	Spain
Iraq	Canada
Emirates	Japan
Bhutan	Argentina

United States, Democracy

A	B
USA	USA
France	Australia
Finland	Ireland
Norway	Canada
UK	UK
Sweden	Iceland
Estonia	Netherlands
Denmark	Austria
Australia	Denmark
Switzerland	Japan
Germany	New Zealand

Australia, Life Expectancy

A	B
Australia	Australia
Ireland	Israel
Iceland	Germany
Andorra	Canada
United States	Iceland
New Zealand	Malta
Austria	Ireland
Belgium	Finland
Canada	United States
Switzerland	Luxembourg
Cyprus	UK

Bangladesh, Life Expectancy

A	B
Bangladesh	Bangladesh
Bhutan	India
Papua NG	Bhutan
India	Myanmar
Gambia	Indonesia
Uganda	Philippines
Nepal	Nepal
Timor-Leste	Pakistan
Pakistan	Mongolia
Mauritania	Viet Nam
Indonesia	Kyrgyzstan

Bulgaria, Life Expectancy

A	B
Bulgaria	Bulgaria
Estonia	Croatia
Portugal	Poland
Macedonia	Serbia
Kuwait	Hungary
Bosnia	Slovakia
Hungary	Bosnia
Croatia	Belarus
Spain	Montenegro
Japan	Estonia
Poland	Montserrat

Japan, Life Expectancy

A	B
Japan	Japan
Hungary	Austria
Portugal	Belgium
Spain	Canada
Slovakia	Switzerland
Greece	Germany
Kuwait	Denmark
Slovenia	Finland
Emirates	France
Poland	UK
Ireland	Netherlands

Qatar, Life Expectancy

A	B
Qatar	Qatar
Emirates	Serbia
Kuwait	Bosnia
Bahrain	Belarus
Turks Isld	Croatia
Cayman Isld	Montenegro
Guernsey	Estonia
Bermuda	Bulgaria
Jersey	Lithuania
Israel	Latvia
Singapore	Saudi Arabia

UK, Life Expectancy

A	B
UK	UK
Belgium	Austria
France	Belgium
Luxembourg	Canada
Slovenia	Switzerland
Germany	Germany
Malta	Denmark
Canada	Finland
Finland	France
Ireland	Japan
Czechia	Netherlands

Hong Kong, Urban Pop

A	B
Hong Kong	Hong Kong
Italy	Singapore
Mexico	Austria
Finland	Canada
Bulgaria	Greenland
Belgium	Netherlands
Lithuania	Andorra
Slovakia	Switzerland
Poland	Ireland
Lebanon	Iceland
Panama	Denmark

Singapore, Urban Pop

A	B
Singapore	Singapore
Barbados	Hong Kong
Oman	Gibraltar
Norway	Andorra
Romania	Monaco
Libya	United States
Algeria	San Marino
Palau	Luxembourg
Gabon	Norway
Cuba	Austria
Switzerland	Australia

Figure 8: The top-10 ranking countries returned by predictive relevance and cosine similarity for each of the 10 queries used for the human evaluation in Figure 7. For each country-context search query, we showed seventy subjects (surveyed on the AI crowdsourcing platform `crowdflower.com`) a pair of tables. We then asked each subject to select the table which contains more relevant results to the search query, or report that both tables contain equally relevant results. The tables above show the top-ranked countries using CrossCat predictive relevance and cosine similarity, with a histogram of the human responses. The caption of Figure 7 describes how we converted these raw histograms into scores between -1 and 1 that are displayed in the main text. The tables showing countries ranked using CrossCat predictive relevance are: Saudi Arabia (A); United States (B); Australia (A); Bangladesh (B); Bulgaria (B); Japan (B); Qatar (A); UK (B); Hong Kong (B); Singapore (B).

C Pairwise heatmaps on Gapminder countries using baseline methods

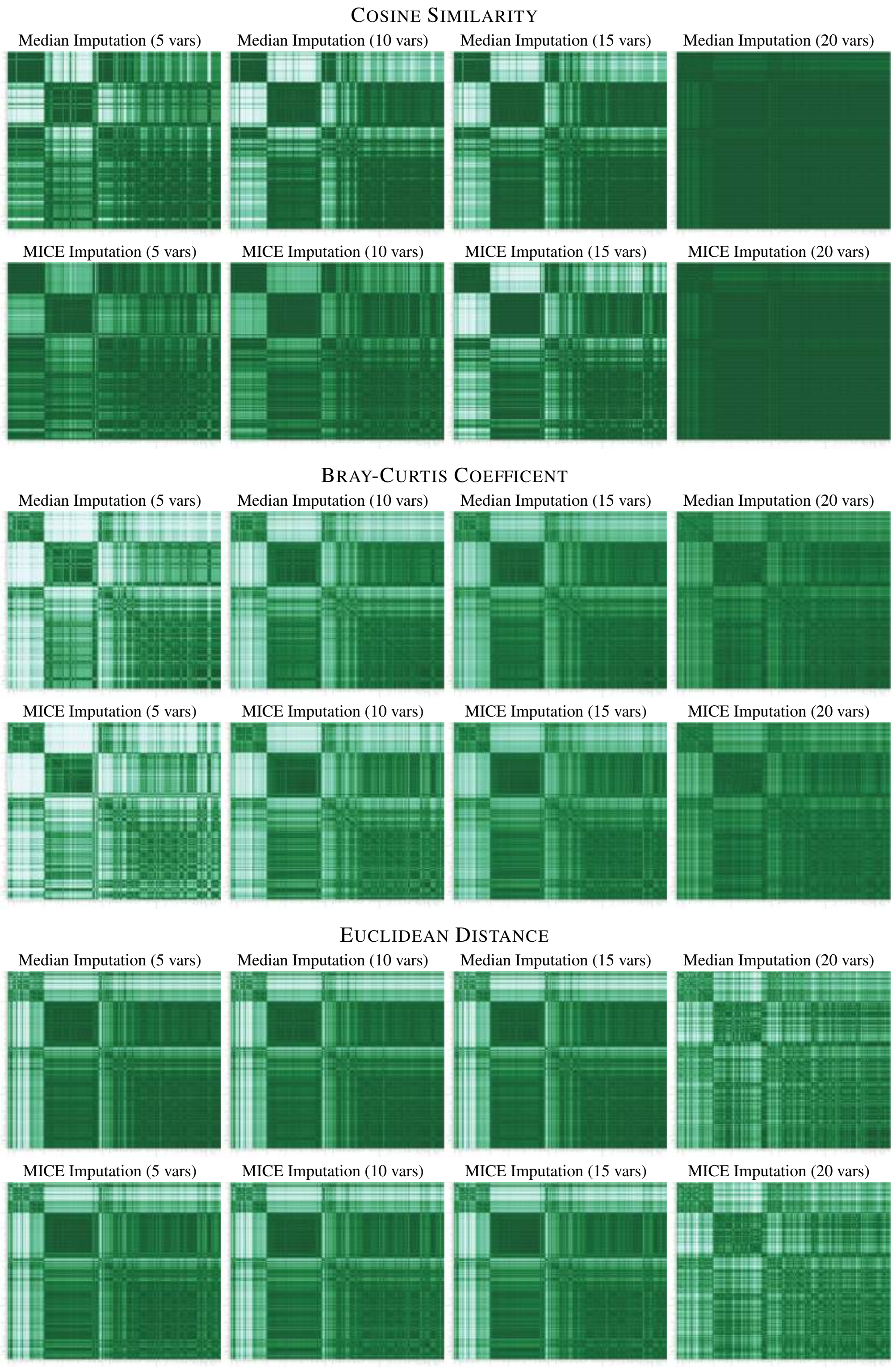


Figure 9: Pairwise heatmaps of countries in Gapminder dataset in the context of "life expectancy at birth", using various distance and similarity measures on the country vectors. Each heatmap is labeled with the imputation technique (median or MICE (Buuren and Groothuis-Oudshoorn, 2011)), and the number of variables in the context (i.e. dimensionality of the vectors). These techniques struggle with sparsity and their structures are much noisier than the results of relevance probability shown in Figure 5a and Table 5e.

D Application to a dataset of 1987 cars

```
%bql CREATE TABLE cars_1987_raw
...  FROM 'cars_1987.csv'
```

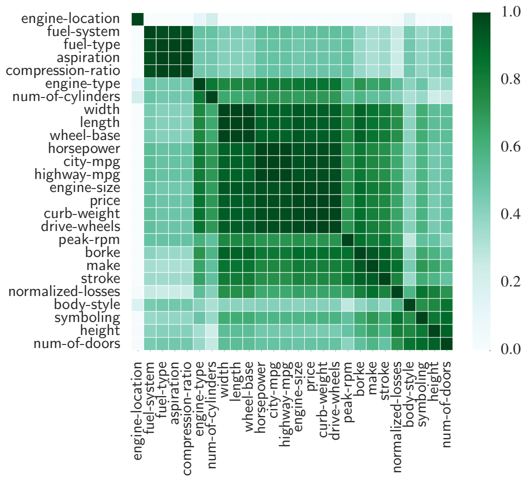
```
%bql SELECT
...   "make",
...   "price",
...   "wheels",
...   "doors",
...   "engine",
...   "horsepower",
...   "body"
...  FROM cars_1987_raw
...  WHERE "price" < 45000
...    AND "wheels" = 'rear'
...    AND "doors" = 'four
...    AND "engine" >= 250
...    AND "horsepower" > 180
...    AND "body" sedan
```

(a) Suppose a customer wishes to purchase a classic car from 1987 with a budget of $45,000 and a desired set of technical specifications. They first load a csv file of 200 cars with 26 variables into a BayesDB table, and then specify the search conditions as Boolean filters in a SQL `WHERE` clause. Due to sparsity in the table, only one record is returned. To obtain more relevant results, the user needs to broaden the specifications in the query.

make	price	wheels	doors	engine	horsepower	body
mercedes	40,960	rear	four	308	184	sedan

```
%mml CREATE POPULATION
...  cars_1987
...  FOR cars_1987_raw
...  WITH SCHEMA (
...    GUESS STATISTICAL
...    TYPES FOR (*);
...  )

%mml CREATE METAMODEL m FOR cars_1987
...  WITH BASELINE crosscat;

%mml INITIALIZE 100 MODELS FOR m;
%mml ANALYZE m FOR 1 MINUTE;

%bql .heatmap ESTIMATE
...  DEPENDENCE PROBABILITY
...  FROM PAIRWISE VARIABLES
...  OF cars_1987
```

(b) Building CrosssCat models in BayesDB for the `cars_1987` population learns a full joint probabilistic model over all variables. The `ESTIMATE DEPENDENCE PROBABILITY` query allows the user to plot a heatmap of probable dependencies between car characteristics. The context of "price" probably contains the majority of other variables in the search query.



```
%bql SELECT
...   "make",
...   "price",
...   "wheels",
...   "doors",
...   "engine-size",
...   "horsepower",
...   "style"
... FROM cars_1987
... ORDER BY
...   RELEVANCE PROBABILITY
...   TO HYPOTHETICAL ROW ((
...    "price" = 42000,
...    "wheels" = 'rear',
...    "doors" = 'four',
...    "engine" = 250,
...    "horsepower" = 180,
...    "body" = 'sedan'
...  ))
...  IN THE CONTEXT OF
...   "price"
... LIMIT 10
```

(c) Using `ORDER BY RELEVANCE PROBABILITY` in BQL ranks each car in the table by its relevance to the user's specifications, which are specified as a hypothetical row. The top-10 ranked cars by probability of relevance to the search query, in the context of `price`, are shown below in the table below. The user can now inspect further characteristics of this subset of cars, to find ones that they like best.

make	price	wheels	doors	engine	horsepower	body
jaguar	35,550	rear	four	258	176	sedan
jaguar	32,250	rear	four	258	176	sedan
mercedes	40,960	rear	four	308	184	sedan
mercedes	45,400	rear	two	304	184	hardtop
mercedes	34,184	rear	four	234	155	sedan
mercedes	35,056	rear	two	234	155	convertible
bmw	36,880	rear	four	209	182	sedan
bmw	41,315	rear	two	209	182	sedan
bmw	30,760	rear	four	209	182	sedan
jaguar	36,000	rear	two	326	262	sedan

Figure 10: A session in BayesDB for probabilistic model building and search in the cars dataset (Kibler et al., 1989).

Detecting Dependencies in Sparse, Multivariate Databases Using Probabilistic Programming and Non-parametric Bayes

Feras Saad
Probabilistic Computing Project
Massachusetts Institute of Technology

Vikash Mansinghka
Probabilistic Computing Project
Massachusetts Institute of Technology

Abstract

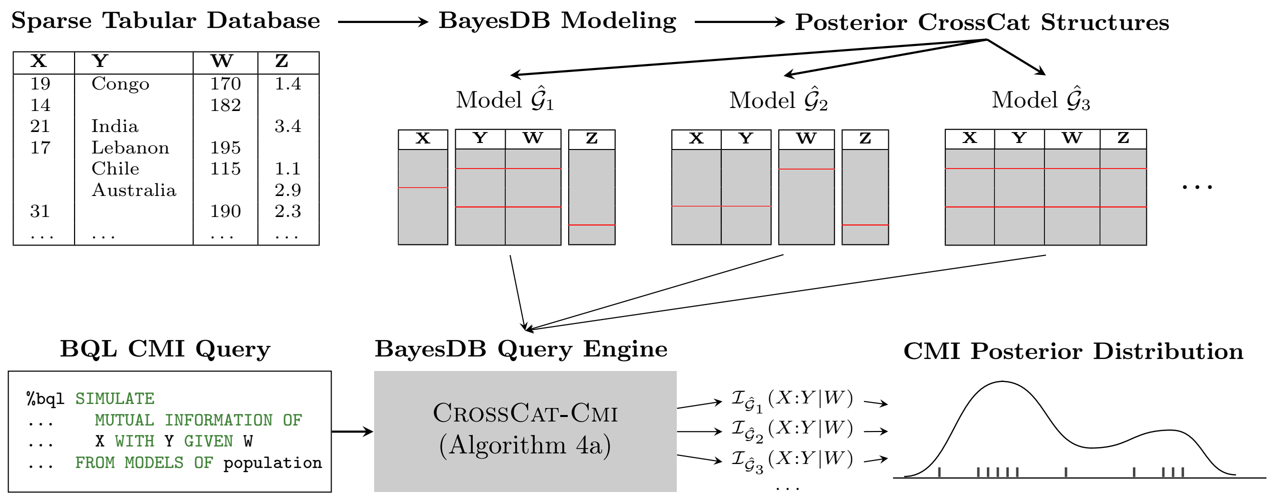
Datasets with hundreds of variables and many missing values are commonplace. In this setting, it is both statistically and computationally challenging to detect true predictive relationships between variables and also to suppress false positives. This paper proposes an approach that combines probabilistic programming, information theory, and non-parametric Bayes. It shows how to use Bayesian non-parametric modeling to (i) build an ensemble of joint probability models for all the variables; (ii) efficiently detect marginal independencies; and (iii) estimate the conditional mutual information between arbitrary subsets of variables, subject to a broad class of constraints. Users can access these capabilities using BayesDB, a probabilistic programming platform for probabilistic data analysis, by writing queries in a simple, SQL-like language. This paper demonstrates empirically that the method can (i) detect context-specific (in)dependencies on challenging synthetic problems and (ii) yield improved sensitivity and specificity over baselines from statistics and machine learning, on a real-world database of over 300 sparsely observed indicators of macroeconomic development and public health.

1 Introduction

Sparse databases with hundreds of variables are commonplace. In these settings, it can be both statistically and computationally challenging to detect predictive relationships between variables [4]. First, the data may be incomplete and require cleaning and imputation before pairwise statistics can be calculated. Second, parametric modeling assumptions that underlie standard hypothesis testing techniques may not be appropriate due to nonlinear, multivariate, and/or heteroskedastic relationships. Third, as the number of variables grows, it becomes harder to detect true relationships while suppressing false positives. Many approaches have been proposed (see [17, Table 1] for a summary), but they each exhibit limitations in practice. For example, some only apply to fully-observed real-valued data, and most do not produce probabilistically coherent measures of uncertainty. This paper proposes an approach to dependence detection that combines probabilistic programming, information theory, and non-parametric Bayes. The end-to-end approach is summarized in Figure 1. Queries about the conditional mutual information (CMI) between variables of interest are expressed using the Bayesian Query Language [18], an SQL-like probabilistic programming language. Approximate inference with CrossCat [19] produces an ensemble of joint probability models, which are analyzed for structural (in)dependencies. For model structures in which dependence cannot be ruled out, the CMI is estimated via Monte Carlo integration.

In principle, this approach has significant advantages. First, the method is scalable to high-dimensional data: it can be used for exploratory analysis without requiring expensive CMI estimation for all pairs of variables. Second, it applies to heterogeneously typed, incomplete datasets with minimal pre-processing [19]. Third, the non-parametric Bayesian joint density estimator used to form CMI estimates can model a broad class of data patterns, without overfitting to highly irregular data. This paper shows that the proposed approach is effective on a real-world database with hundreds of variables and a missing data rate of ~35%, detecting common-sense predictive relationships that are missed by baseline methods while suppressing spurious relationships that baselines purport to detect.

Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).


Sparse Tabular Database
BayesDB Modeling
Posterior CrossCat Structures
Model $\hat{\mathcal{G}}_1$
Model $\hat{\mathcal{G}}_2$
Model $\hat{\mathcal{G}}_3$
X
Y
W
Z
BQL CMI Query
BayesDB Query Engine
CMI Posterior Distribution
CrossCat-Cmi (Algorithm 4a)
$\mathcal{I}_{\hat{\mathcal{G}}_1}(X{:}Y|W)$
$\mathcal{I}_{\hat{\mathcal{G}}_2}(X{:}Y|W)$
$\mathcal{I}_{\hat{\mathcal{G}}_3}(X{:}Y|W)$

X	Y	W	Z
19	Congo	170	1.4
14		182	
21	India		3.4
17	Lebanon	195	
	Chile	115	1.1
	Australia		2.9
31		190	2.3
...	...	...	...

```
%bql SIMULATE
...     MUTUAL INFORMATION OF
...     X WITH Y GIVEN W
... FROM MODELS OF population
```

Figure 1: Workflow for computing posterior distributions of the CMI for variables in a data table using BayesDB. Modeling and inference in BayesDB produces an ensemble of posterior CrossCat samples. Each model learns a factorization of the joint distribution of all variables in the database, and a Dirichlet process mixture within each block of dependent variables. For instance, model $\hat{\mathcal{G}}_1$ specifies that X is independent of (Y, W) which in turn is independent of Z, while in $\hat{\mathcal{G}}_3$, all variables are (structurally) dependent. End-user queries for the CMI are expressed in the Bayesian Query Language. The BQL interpreter uses CrossCat structures to optimize the query where possible, by (i) bypassing Monte Carlo estimation completely when the queried variables are structurally independent, and/or (ii) dropping redundant constraints which are structurally independent of the queried variables. Values of CMI returned by each model constitute samples from the posterior CMI distribution.

2 Drawing Bayesian inferences about conditional mutual information

Let $\boldsymbol{x} = (x_1, x_2, \dots, x_D)$ denote a D-dimensional random vector, whose sub-vectors we denote $\boldsymbol{x}_\mathcal{A} = \{x_i : i \in \mathcal{A}\}$ with joint probability density $p_\mathcal{G}(\boldsymbol{x}_\mathcal{A})$. The symbol $\mathcal{G}$ refers to an arbitrary specification for the "generative" process of $\boldsymbol{x}$, and parameterizes all its joint and conditional densities. The *mutual information* (MI) of the variables $\boldsymbol{x}_\mathcal{A}$ and $\boldsymbol{x}_\mathcal{B}$ (under generative process $\mathcal{G}$) is defined in the usual way [5]:

$$\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}) = \mathop{\mathbb{E}}_{(\boldsymbol{x}_\mathcal{A}, \boldsymbol{x}_\mathcal{B})} \left[\log \left(\frac{p_\mathcal{G}(\boldsymbol{x}_\mathcal{A}, \boldsymbol{x}_\mathcal{B})}{p_\mathcal{G}(\boldsymbol{x}_\mathcal{A}) p_\mathcal{G}(\boldsymbol{x}_\mathcal{B})} \right) \right]. \quad (1)$$

The mutual information can be interpreted as the KL-divergence from the product of marginals $p_\mathcal{G}(\boldsymbol{x}_\mathcal{A}) p_\mathcal{G}(\boldsymbol{x}_\mathcal{B})$ to the joint distribution $p_\mathcal{G}(\boldsymbol{x}_\mathcal{A}, \boldsymbol{x}_\mathcal{B})$, and is a well-established measure for both the existence and strength of dependence between $\boldsymbol{x}_\mathcal{A}$ and $\boldsymbol{x}_\mathcal{B}$ (Section 2.2). Given an observation of the variables $\{\boldsymbol{x}_\mathcal{C} {=} \hat{\boldsymbol{x}}_\mathcal{C}\}$, the *conditional mutual information* (CMI) of $\boldsymbol{x}_\mathcal{A}$ and $\boldsymbol{x}_\mathcal{B}$ given $\{\boldsymbol{x}_\mathcal{C} {=} \hat{\boldsymbol{x}}_\mathcal{C}\}$ is defined analogously:

$$\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B} | \boldsymbol{x}_\mathcal{C} {=} \hat{\boldsymbol{x}}_\mathcal{C}) = \mathop{\mathbb{E}}_{(\boldsymbol{x}_\mathcal{A}, \boldsymbol{x}_\mathcal{B}) | \hat{\boldsymbol{x}}_\mathcal{C}} \left[\log \left(\frac{p_\mathcal{G}(\boldsymbol{x}_\mathcal{A}, \boldsymbol{x}_\mathcal{B} | \hat{\boldsymbol{x}}_\mathcal{C})}{p_\mathcal{G}(\boldsymbol{x}_\mathcal{A} | \hat{\boldsymbol{x}}_\mathcal{C}) p_\mathcal{G}(\boldsymbol{x}_\mathcal{B} | \hat{\boldsymbol{x}}_\mathcal{C})} \right) \right]. \quad (2)$$

Estimating the mutual information between the variables of $\boldsymbol{x}$ given a dataset of observations $\mathcal{D}$ remains an open problem in the literature. Various parametric and non-parametric methods for estimating MI exist [21, 22, 15]; see [24] for a comprehensive review. Traditional approaches typically construct a point estimate $\hat{\mathcal{I}}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B})$ (and possible confidence intervals) assuming a "true value" of $\mathcal{I}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B})$. In this paper, we instead take a non-parametric Bayesian approach, where the mutual information itself is a derived random variable; a similar interpretation was recently developed in independent work [16]. The randomness of mutual information arises from treating the data generating process and parameters $\mathcal{G}$ as a random variable, whose prior distribution we denote π. Composing $\mathcal{G}$ with the function $h : \hat{\mathcal{G}} \mapsto \mathcal{I}_{\hat{\mathcal{G}}}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B})$ induces the derived random variable $h(\mathcal{G}) \equiv \mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B})$. The distribution of the MI can thus be expressed as an expectation under distribution π:

$$\begin{aligned} \mathbb{P}\left[\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}) \in S\right] &= \int \mathbb{I}\left[\mathcal{I}_{\hat{\mathcal{G}}}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}) \in S\right] \pi(d\hat{\mathcal{G}}) \\ &= \mathop{\mathbb{E}}_{\hat{\mathcal{G}} \sim \pi} \left[\mathbb{I}\left[\mathcal{I}_{\hat{\mathcal{G}}}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}) \in S\right]\right]. \quad (3) \end{aligned}$$

Given a dataset $\mathcal{D}$, we define the posterior distribution of the mutual information, $\mathbb{P}\left[\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}) \in S | \mathcal{D}\right]$ as the expectation in Eq (3) under the posterior $\pi(\cdot|\mathcal{D})$. We define the distribution over conditional mutual information $\mathbb{P}\left[\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B} | \hat{\boldsymbol{x}}_\mathcal{C}) \in S\right]$ analogously to Eq (3), substituting the CMI (2) inside the expectation.

2.1 Estimating CMI with generative population models

Monte Carlo estimates of CMI can be formed for models expressed as *generative population models* [18, 28], a probabilistic programming formalism for characterizing the data generating process of an infinite array of realizations of random vector $\boldsymbol{x} = (x_1, x_2, \dots, x_D)$. Listing 1 summarizes elements of the GPM interface.

Listing 1 GPM interface for simulating from and assessing the density of conditional and marginal distributions of a random vector $\boldsymbol{x}$.

```
SIMULATE($\mathcal{G}$, query: $\mathcal{Q}=\{q_j\}$, condition: $\hat{\boldsymbol{x}}_\mathcal{E}=\{\hat{x}_{e_j}\}$)
  Return a sample $\mathbf{s} \sim p_\mathcal{G}(\boldsymbol{x}_\mathcal{Q}|\hat{\boldsymbol{x}}_\mathcal{E}, \mathcal{D})$.

LOGPDF($\mathcal{G}$, query: $\hat{\boldsymbol{x}}_\mathcal{Q}=\{\hat{x}_{q_j}\}$, condition: $\hat{\boldsymbol{x}}_\mathcal{E}=\{\hat{x}_{e_j}\}$)
  Return the joint log density $p_\mathcal{G}(\hat{\boldsymbol{x}}_\mathcal{Q}|\hat{\boldsymbol{x}}_\mathcal{E}, \mathcal{D})$
```

These two interface procedures can be combined to derive a simple Monte Carlo estimator for the CMI (2), shown in Algorithm 2a.

Algorithm 2a GPM-CMI

```
Require: GPM $\mathcal{G}$; query $\mathcal{A}$, $\mathcal{B}$; condition $\hat{\boldsymbol{x}}_\mathcal{C}$; accuracy $T$
Ensure: Monte Carlo estimate of $\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}|\boldsymbol{x}_\mathcal{C}{=}\hat{\boldsymbol{x}}_\mathcal{C})$
1: for $t = 1, \dots, T$ do
2:     $(\hat{\boldsymbol{x}}_\mathcal{A}, \hat{\boldsymbol{x}}_\mathcal{B}) \leftarrow$ SIMULATE$(\mathcal{G}, \mathcal{A} \cup \mathcal{B}, \hat{\boldsymbol{x}}_\mathcal{C})$
3:     $m^t_{\mathcal{A}\cup\mathcal{B}} \leftarrow$ LOGPDF$(\mathcal{G}, \hat{\boldsymbol{x}}_{\mathcal{A}\cup\mathcal{B}}, \hat{\boldsymbol{x}}_\mathcal{C})$
4:     $m^t_\mathcal{A} \leftarrow$ LOGPDF$(\hat{\boldsymbol{x}}_\mathcal{A}, \hat{\boldsymbol{x}}_\mathcal{C})$
5:     $m^t_\mathcal{B} \leftarrow$ LOGPDF$(\hat{\boldsymbol{x}}_\mathcal{B}, \hat{\boldsymbol{x}}_\mathcal{C})$
6: return $\frac{1}{T}\sum_{t=1}^{T}\left(m^t_{\mathcal{A}\cup\mathcal{B}} - (m^t_\mathcal{A} + m^t_\mathcal{B})\right)$
```

While GPM-CMI is an unbiased and consistent estimator applicable to any probabilistic model implemented as a GPM, its quality in detecting dependencies is tied to the ability of $\mathcal{G}$ to capture patterns from the dataset $\mathcal{D}$; this paper uses baseline non-parametric GPMs built using CrossCat (Section 3).

2.2 Extracting conditional independence relationships from CMI estimates

An estimator for the CMI can be used to discover several forms of independence relations of interest.

Marginal Independence It is straightforward to see that $(\boldsymbol{x}_\mathcal{A} \perp\!\!\!\perp_\mathcal{G} \boldsymbol{x}_\mathcal{B})$ if and only if $\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}) = 0$.

Context-Specific Independence If the event $\{\boldsymbol{x}_\mathcal{C}{=}\hat{\boldsymbol{x}}_\mathcal{C}\}$ decouples $\boldsymbol{x}_\mathcal{A}$ and $\boldsymbol{x}_\mathcal{B}$, then they are said to be independent "in the context" of $\hat{\boldsymbol{x}}_\mathcal{C}$, denoted $(\boldsymbol{x}_\mathcal{A} \perp\!\!\!\perp_\mathcal{G} \boldsymbol{x}_\mathcal{B}|\, \{\boldsymbol{x}_\mathcal{C}{=}\hat{\boldsymbol{x}}_\mathcal{C}\})$ [3]. This condition is equivalent to the CMI from (2) equaling zero. Thus by estimating CMI, we are able to detect finer-grained independencies than can be detected by analyzing the graph structure of a learned Bayesian network [30].

Conditional Independence If context-specific independence holds for all possible observation sets $\{\boldsymbol{x}_\mathcal{C}{=}\hat{\boldsymbol{x}}_\mathcal{C}\}$, then $\boldsymbol{x}_\mathcal{A}$ and $\boldsymbol{x}_\mathcal{B}$ are *conditionally independent* given $\boldsymbol{x}_\mathcal{C}$, denoted $(\boldsymbol{x}_\mathcal{A} \perp\!\!\!\perp_\mathcal{G} \boldsymbol{x}_\mathcal{B}|\boldsymbol{x}_\mathcal{C})$. By the non-negativity of CMI, conditional independence implies the CMI of $\boldsymbol{x}_\mathcal{A}$ and $\boldsymbol{x}_\mathcal{B}$, marginalizing out $\boldsymbol{x}_\mathcal{C}$, is zero:

$$\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}|\boldsymbol{x}_\mathcal{C}) = \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}_\mathcal{C}}\left[\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}|\boldsymbol{x}_\mathcal{C} = \hat{\boldsymbol{x}}_\mathcal{C})\right] = 0. \quad (4)$$

Figure 2 illustrates different CMI queries which are used to discover these three types of dependencies in various data generators; Figure 3 shows CMI queries expressed in the Bayesian Query Language.

3 Building generative population models for CMI estimation with non-parametric Bayes

Our approach to estimating the CMI requires a prior π and model class $\mathcal{G}$ which is flexible enough to emulate an arbitrary joint distribution over $\boldsymbol{x}$, and tractable enough to implement Algorithm 2a for its arbitrary sub-vectors. We begin with a Dirichlet process mixture model (DPMM) [12]. Letting L_d denote the likelihood for variable d, V_d a prior over the parameters of L_d, and $\boldsymbol{\lambda}_d$ the hyperparameters of V_d, the generative process for N observations $\mathcal{D}=\left\{\boldsymbol{x}_{[i,1:D]} : 1 \le i \le N\right\}$ is:

<u>DPMM-PRIOR</u>

$$\begin{aligned}
&\alpha \sim \text{GAMMA}(1,1) \\
&\boldsymbol{z} = (z_1, \dots, z_N) \sim \text{CRP}(\cdot|\alpha) \\
&\boldsymbol{\phi}_{[d,k]} \sim V_d(\cdot|\boldsymbol{\lambda}_d) && d \in [D], k \in \text{UNIQUE}(\boldsymbol{z}) \\
&x_{[i,d]} \sim L_d(\cdot|\boldsymbol{\phi}_{[d,z_i]}) && i \in [N], d \in [D]
\end{aligned}$$

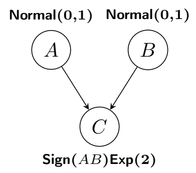
We refer to [7, 14] for algorithms for posterior inference, and assume we have a posterior sample $\hat{\mathcal{G}} = (\alpha, \boldsymbol{z}_{[1:N]}, \{\boldsymbol{\phi}_d\})$ of all parameters in the DPMM. To compute the CMI of an arbitrary query pattern $\mathcal{I}_{\hat{\mathcal{G}}}(\boldsymbol{x}_\mathcal{A}{:}\boldsymbol{x}_\mathcal{B}|\boldsymbol{x}_\mathcal{C}{=}\hat{\boldsymbol{x}}_\mathcal{C})$ using Algorithm 2a, we need implementations of SIMULATE and LOGPDF for $\hat{\mathcal{G}}$. These two procedures are summarized in Algorithms 3a, 3b.

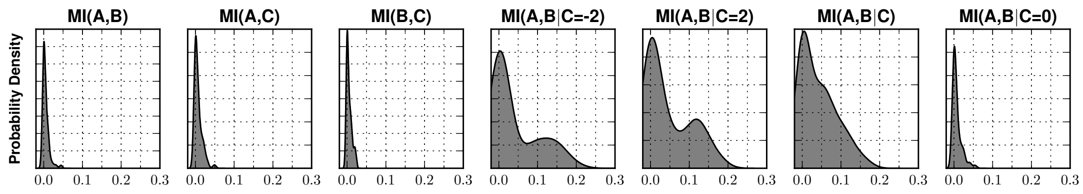
Algorithm 3a DPMM-SIMULATE

```
Require: DPMM $\mathcal{G}$; target $\mathcal{A}$; condition $\hat{\boldsymbol{x}}_\mathcal{C}$
Ensure: joint sample $\hat{\boldsymbol{x}}_\mathcal{A} \sim p_\mathcal{G}(\cdot|\hat{\boldsymbol{x}}_\mathcal{C})$
1: $(l_i)_{i=1}^{K+1} \leftarrow$ DPMM-CLUSTER-POSTERIOR$(\mathcal{G}, \hat{\boldsymbol{x}}_\mathcal{C})$
2: $z_{N+1} \sim$ CATEGORICAL$(l_1, \dots, l_{K+1})$
3: for $a \in \mathcal{A}$ do
4:     $\hat{x}_a \sim L_a(\cdot|\boldsymbol{\phi}_{[a,z_{N+1}]})$
5: return $\hat{\boldsymbol{x}}_\mathcal{A}$
```

Algorithm 3b DPMM-LOGPDF

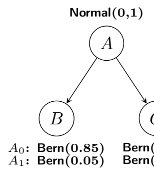
```
Require: DPMM $\mathcal{G}$; target $\hat{\boldsymbol{x}}_\mathcal{A}$; condition $\hat{\boldsymbol{x}}_\mathcal{C}$
Ensure: log density $p_\mathcal{G}(\hat{\boldsymbol{x}}_\mathcal{A}|\hat{\boldsymbol{x}}_\mathcal{C})$
1: $(l_i)_{i=1}^{K+1} \leftarrow$ DPMM-CLUSTER-POSTERIOR$(\mathcal{G}, \hat{\boldsymbol{x}}_\mathcal{C})$
2: for $k = 1, \dots, K+1$ do
3:     $t_k \leftarrow \prod_{a\in\mathcal{A}} L_a(\hat{x}_a|\boldsymbol{\phi}_{[a,k]})$
4: return $\log\left(\sum_{k=1}^{K+1}(t_k l_k)\right)$
```

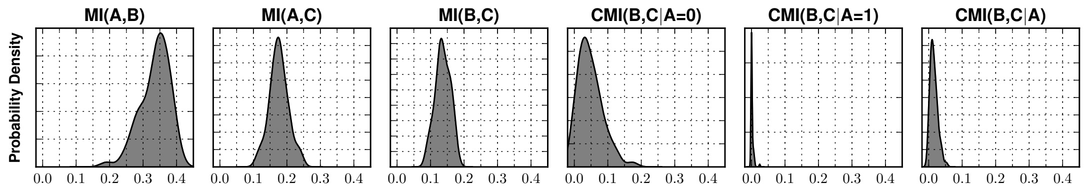




(a) Ground truth "common-effect" generator [29].

(b) The first three plots verify that A, B, and C are marginally independent. The next three plots show that conditioning on C "couples" the parents A and B (both for fixed values of $C \in \{2, -2\}$, and marginalizing over all C). The last plot shows that $\{C = 0\}$ does not couple A and B, due to symmetry of signum.





(c) Ground truth "common-cause" generator.

(d) The first three plots verify that A, B, and C are marginally dependent. The next three plots show that conditioning on A "decouples" the children B and C; the decoupling is weaker for $\{A = 0\}$, because it is 3.4 nats less likely that $\{A = 1\}$. The final plot shows the weighted CMI under these two possibilities.

Figure 2: Posterior distributions of CMI under the DPMM posterior, given 100 data points from canonical Bayes net structures. Distributions peaked at 0 indicate high probability of (conditional) independence. In both cases, the posterior CMI distributions correctly detect the marginal, conditional, and context-specific independences in the "ground truth" Bayes nets, despite the fact that both "common-cause" and "common-effect" structures are not in the (structural) hypothesis space of the DPMM prior.

Algorithm 3c DPMM-Cluster-Posterior

```
Require: DPMM $\mathcal{G}$; condition $\hat{\boldsymbol{x}}_{\mathcal{C}}$;
Ensure: $\{p_{\mathcal{G}}(z_{N+1} = k) : 1 \le k \le \max(\boldsymbol{z}_{1:N}) + 1\}$
1: $K \leftarrow \max(\boldsymbol{z}_{1:N})$
2: for $k = 1, \dots, K+1$ do
3:     $n_k \leftarrow \begin{cases} |\{\boldsymbol{x}_i \in \mathcal{D} : z_i = k\}| & \text{if } k \le K \\ \alpha & \text{if } k = K+1 \end{cases}$
4:     $l_k \leftarrow \left(\prod_{c \in \mathcal{C}} L_c(\hat{x}_c | \boldsymbol{\phi}_{[c,k]})\right) n_k$
5: return $(l_1, \dots, l_{K+1}) / \sum_{k=1}^{K+1}(l_k)$
```

The subroutine DPMM-Cluster-Posterior is used for sampling (in DPMM-Simulate) and marginalizing over (in DPMM-LogPdf) the non-parametric mixture components. Moreover, if L_d and V_d form a conjugate likelihood-prior pair, then invocations of $L_d(\hat{x}_d | \boldsymbol{\phi}_{[d,k]})$ in Algorithms 3a:4 and 3b:3 can be Rao-Blackwellized by conditioning on the sufficient statistics of data in cluster k, thus marginalizing out $\boldsymbol{\phi}_{[d,k]}$ [26]. This optimization is important in practice, since analytical marginalization can be obtained in closed-form for several likelihoods in the exponential family [9]. Finally, to approximate the posterior distribution over CMI in (2), it suffices to aggregate DPMM-Cmi from a set of posterior samples $\left\{\hat{\mathcal{G}}_1, \dots, \hat{\mathcal{G}}_H\right\} \overset{\text{iid}}{\sim} \pi(\cdot|\mathcal{D})$. Figure 2 shows posterior CMI distributions from the DPMM successfully recovering the marginal and conditional independencies in two canonical Bayesian networks.

3.1 Inducing sparse dependencies using the CrossCat prior

The multivariate DPMM makes the restrictive assumption that all variables $\boldsymbol{x} = (x_1, \dots, x_D)$ are (structurally) marginally dependent, where their joint distribution fully factorizes conditioned on the mixture assignment z. In high-dimensional datasets, imposing full structural dependence among all variables is too conservative. Moreover, while the Monte Carlo error of Algorithm 2a does not scale with the dimensionality D, its runtime scales linearly for the DPMM, and so estimating the CMI is likely to be prohibitively expensive. We relax these constraints by using CrossCat [19], a structure learning prior which induces sparsity over the dependencies between the variables of $\boldsymbol{x}$. In particular, CrossCat posits a factorization of $\boldsymbol{x}$ according to a *variable partition* $\boldsymbol{\gamma} = \{\mathcal{V}_1, \dots, \mathcal{V}_{|\boldsymbol{\gamma}|}\}$, where $\mathcal{V}_i \subseteq [D]$. For $i \ne j$, all variables in block $\mathcal{V}_i$ are mutually (marginally and conditionally) independent of all variables in $\mathcal{V}_j$. The factorization of $\boldsymbol{x}$ given the

English Summary of CMI Query	**CMI Query in Bayesian Query Language**	**Inference Algorithm Invoked by Query Interpreter**
Simulate from the posterior distribution of the mutual information of (x_1, x_2) with x_3, given $x_4 = 14$.	`SIMULATE` `MUTUAL INFORMATION OF` `(x1, x2) WITH (x3)` `GIVEN (x4 = 14)` `FROM MODELS OF population`	1: **for** $\mathcal{G}_k \in \mathcal{M}$ **do** 2: $\mathcal{I}_{\mathcal{G}_k} \leftarrow$ GPM-CMI$(\mathcal{G}_k, \{x_1, x_2\}, \{x_3\}, \{(x_4, 14)\})$ 3: **return** $(\mathcal{I}_{\mathcal{G}_1}, \dots, \mathcal{I}_{\mathcal{G}_{\lvert\mathcal{M}\rvert}})$
Estimate the probability that the mutual information of (x_1, x_2) with x_3, given $x_4 = 14$ and marginalizing over x_5, is less than 0.1 nats.	`ESTIMATE PROBABILITY OF` `MUTUAL INFORMATION OF` `(x1, x2) WITH (x3)` `GIVEN (x4 = 14, x5)` `< 0.1` `BY population`	1: **for** $\mathcal{G}_k \in \mathcal{M}$ **do** 2: **for** $t = 1, \dots, T$ **do** 3: $\hat{x}_5^t \leftarrow$ SIMULATE$(\mathcal{G}_k, x_5, \{(x_4, 14)\})$ 4: $\mathcal{I}^t_{\mathcal{G}_k} \leftarrow$ GPM-CMI$(\mathcal{G}_k, \{x_1, x_2\}, \{x_3\}, \{(x_4, 14), (x_5, \hat{x}_5^t)\})$ 5: $\mathcal{I}_{\mathcal{G}_k} \leftarrow \frac{1}{T}\sum_t \left(\mathcal{I}^t_{\mathcal{G}_k}\right)$ 6: **return** $\frac{1}{\lvert\mathcal{M}\rvert}\sum_j \left(\mathbb{I}\left[\mathcal{I}_{\mathcal{G}_k} < 0.1\right]\right)$
Synthesize a hypothetical dataset with 100 records, including only those variables which are probably independent of x_2.	`SIMULATE (` `SELECT * FROM` `VARIABLES OF population` `WHERE PROBABILITY OF` `MUTUAL INFORMATION` `WITH x2 < 0.1` `> 0.9)` `FROM population` `LIMIT 100;`	1: $\mathcal{S} \leftarrow \varnothing$ 2: **for** $x_i \in (x_1, \dots, x_D)$ **do** 3: **for** $\mathcal{G}_k \in \mathcal{M}$ **do** 4: $\mathcal{I}_{\mathcal{G}_k} \leftarrow$ GPM-CMI$(\mathcal{G}_k, x_i, x_2, \varnothing)$ 5: $p_i \leftarrow \frac{1}{\lvert\mathcal{M}\rvert}\sum_k \mathbb{I}\left[\mathcal{I}_{\mathcal{G}_k} < 0.1\right]$ 6: **if** $p_i > 0.9$ **then** 7: $\mathcal{S} \leftarrow \mathcal{S} \cup \{x_i\}$ 8: **for** $t = 1 \dots, 100$ **do** 9: $s_t \leftarrow$ SIMULATE$(\mathcal{M}, \mathcal{S}, \varnothing)$ 10: **return** $(s_1, \dots, s_{100})$

Figure 3: End-user CMI queries in the Bayesian Query Language for three data analysis tasks; (top) evaluating the strength of predictive relationships; (middle) specifying the amount of evidence required for a "predictively" significant relationship; (bottom) synthesizing a hypothetical population, censoring probably sensitive variables.

variable partition $\boldsymbol{\gamma}$ is therefore given by:

$$p_{\mathcal{G}}(\boldsymbol{x}|\mathcal{D}) = \prod_{\mathcal{V} \in \boldsymbol{\gamma}} p_{\mathcal{G}_\mathcal{V}}(\boldsymbol{x}_\mathcal{V}|\mathcal{D}_\mathcal{V}). \qquad (5)$$

Within block $\mathcal{V}$, the variables $\boldsymbol{x}_\mathcal{V} = \{x_d : d \in \mathcal{V}\}$ are distributed according to a multivariate DPMM; subscripts with $\mathcal{V}$ (such as $\mathcal{G}_\mathcal{V}$) now index a set of block-specific DPMM parameters. The joint predictive density $p_{\mathcal{G}_\mathcal{V}}$ is given by Algorithm 3b:

$$p_{\mathcal{G}_\mathcal{V}}(\boldsymbol{x}_\mathcal{V}|\mathcal{D}) = \sum_{k=1}^{K_V+1} \left(\frac{n_{[\mathcal{V},k]} \prod_{d \in \mathcal{V}} p_{\mathcal{G}_\mathcal{V}}(x_d|\boldsymbol{\phi}_{[d,k]})}{\sum_{k'} n_{[\mathcal{V},k']}} \right). \qquad (6)$$

The CrossCat generative process for N observations $\mathcal{D} = \left\{\boldsymbol{x}_{[i,1:D]} : 1 \le i \le N\right\}$ is summarized below.

<u>CROSSCAT-PRIOR</u>

$\alpha' \sim \text{GAMMA}(1, 1)$

$\boldsymbol{v} = (v_1, \dots, v_D) \sim \text{CRP}(\cdot|\alpha')$

$\mathcal{V}_k \leftarrow \{i \in [D] : v_i = k\} \qquad k \in \text{UNIQUE}(\boldsymbol{v})$

$\left\{x_{[i,\mathcal{V}_k]}\right\}_{i=1}^N \sim \text{DPMM-PRIOR} \qquad k \in \text{UNIQUE}(\boldsymbol{v})$

We refer to [19, 23] for algorithms for posterior inference in CrossCat, and assume we have a set of approximate samples $\left\{\hat{\mathcal{G}}_i : 1 \le i \le H\right\}$ of all latent CrossCat parameters from the posterior $\pi(\cdot|\mathcal{D})$.

3.2 Optimizing a CMI query

The following lemma shows how CrossCat induces sparsity for a multivariate CMI query.

Lemma 1. *Let $\mathcal{G}$ be a posterior sample from CrossCat, whose full joint distribution is given by* (5) *and* (6)*. Then, for all $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq [D]$,*

$$\mathcal{I}_{\mathcal{G}}\left(\boldsymbol{x}_\mathcal{A} : \boldsymbol{x}_\mathcal{B} \mid \hat{\boldsymbol{x}}_\mathcal{C}\right) = \sum_{\mathcal{V} \in \boldsymbol{\gamma}} \mathcal{I}_{\mathcal{G}_\mathcal{V}}(\boldsymbol{x}_{\mathcal{A}\cap\mathcal{V}} : \boldsymbol{x}_{\mathcal{B}\cap\mathcal{V}} \mid \hat{\boldsymbol{x}}_{\mathcal{C}\cap\mathcal{V}}),$$

where $\mathcal{I}_{\mathcal{G}_\mathcal{V}}(\boldsymbol{x}_{\mathcal{A}\cap\mathcal{V}} : \varnothing | \hat{\boldsymbol{x}}_{\mathcal{C}\cap\mathcal{V}}) \equiv 0$.

Proof. Refer to Appendix A.

An immediate consequence of Lemma 1 is that structure discovery in CrossCat allows us to optimize Monte Carlo estimation of $\mathcal{I}_{\mathcal{G}}(\boldsymbol{x}_\mathcal{A} : \boldsymbol{x}_\mathcal{B} | \boldsymbol{x}_\mathcal{C} = \hat{\boldsymbol{x}}_\mathcal{C})$ by ignoring all target and condition variables which are not in the same block $\mathcal{V}$, as shown in Algorithm 4a and Figure 4.

Algorithm 4a CROSSCAT-CMI

```
Require: CrossCat $\mathcal{G}$; query $\mathcal{A}$, $\mathcal{B}$; condition $\hat{\boldsymbol{x}}_\mathcal{C}$; acc. $T$
Ensure: Monte Carlo estimate of $\mathcal{I}_\mathcal{G}(\boldsymbol{x}_\mathcal{A}:\boldsymbol{x}_\mathcal{B}|\boldsymbol{x}_\mathcal{C}=\hat{\boldsymbol{x}}_\mathcal{C})$
1: for $\mathcal{V} \in \boldsymbol{\gamma}$ do
2:   if $\mathcal{A} \cap \mathcal{V}$ AND $\mathcal{B} \cap \mathcal{V}$ then
3:     $i_\mathcal{V} \leftarrow$ GPM-CMI($\mathcal{G}_\mathcal{V}$, $\mathcal{A} \cap \mathcal{V}$, $\mathcal{B} \cap \mathcal{V}$, $\hat{\boldsymbol{x}}_{\mathcal{C}\cap\mathcal{V}}$, $T$)
4:   else
5:     $i_\mathcal{V} \leftarrow 0$
6: return $\sum_{\mathcal{V}\in\boldsymbol{\gamma}} i_\mathcal{V}$
```

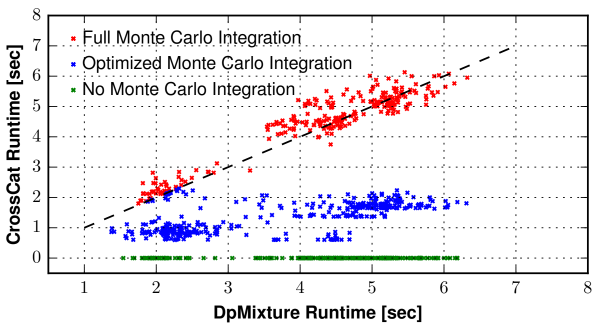


Figure 4: Comparing the runtime of CrossCat-Cmi (Alg 4a) and Gpm-Cmi (Alg 2a) (using the DPMM), on 1000 randomly generated CMI queries from an 8-dimensional dataset. The dashed curve shows the 45-degree line. The green dots at 0 correspond to CrossCat detecting structural independence between query variables, bypassing Monte Carlo estimation completely. The blue dots (below diagonal) correspond to CrossCat optimizing the Monte Carlo estimator by ignoring constraint variables which are structurally independent of the target variables. The red dots (along diagonal) correspond to CrossCat learning no structural independences, requiring full Monte Carlo estimation and resulting in comparable runtime to DPMM. These three cases correspond to the three posterior CrossCat structures illustrated in Figure 1, when the targets variables are X and Y conditioned on W.

3.3 Upper bounding the pairwise dependence probability

In exploratory data analysis, we are often interested in detecting pairwise predictive relationships between variables (x_i, x_j). Using the formalism from Eq (3), we can compute the probability that their MI is non-zero: $\mathbb{P}\left[\mathcal{I}_{\mathcal{G}}(x_i : x_j) > 0\right]$. This quantity can be upper-bounded by the posterior probability that x_i and x_j have the same assignments v_i and v_j in the CrossCat variable partition $\boldsymbol{\gamma}$:

$$\begin{aligned}
&\mathbb{P}\left[\mathcal{I}_{\mathcal{G}}(x_i : x_j) > 0\right] \\
&= \mathbb{P}\left[\mathcal{I}_{\mathcal{G}}(x_i : x_j) > 0 \mid \{\mathcal{G} : v_i = v_j\}\right] \mathbb{P}\left[\{\mathcal{G} : v_i = v_j\}\right] \\
&\quad + \mathbb{P}\left[\mathcal{I}_{\mathcal{G}}(x_i : x_j) > 0 \mid \{\mathcal{G} : v_i \neq v_j\}\right] \mathbb{P}\left[\{\mathcal{G} : v_i \neq v_j\}\right] \\
&= \mathbb{P}\left[\mathcal{I}_{\mathcal{G}}(x_i : x_j) > 0 \mid \{\mathcal{G} : v_i = v_j\}\right] \mathbb{P}\left[\{\mathcal{G} : v_i = v_j\}\right] \\
&< \mathbb{P}\left[\{\mathcal{G} : v_i = v_j\}\right] \approx \frac{1}{H} \sum_{h=1}^{H} \mathbb{I}[\hat{\mathcal{G}}_h : \hat{v}_{[h,i]} = \hat{v}_{[h,j]}], \quad (7)
\end{aligned}$$

where Lemma 1 has been used to set the addend in line 3 to zero. Also note that the summand in (7) can be computed in $O(1)$ for CrossCat sample $\hat{\mathcal{G}}_h$. When dependencies among the D variables are sparse such that many pairs (x_i, x_j) have MI upper bounded by 0, the number of invocations of Algorithm 4a required to compute pairwise MI values is $\ll O(D^2)$. A comparison of upper bounding MI versus exact MI estimation with Monte Carlo is shown in Figure 5.

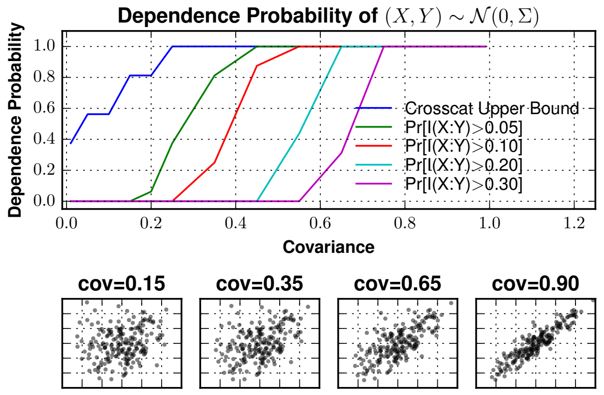
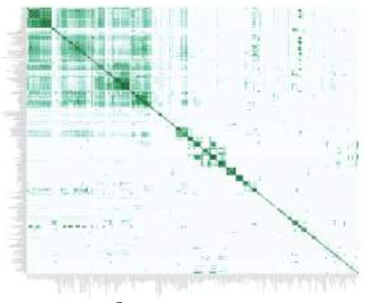


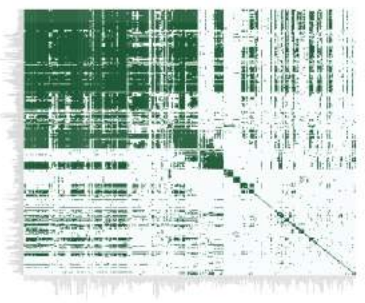
Figure 5: Posterior probability that dimensions of a bivariate Gaussian are dependent, vs the covariance (top). The CrossCat upper bound (7) is useful for detecting the existence of a predictive relationship; the posterior distribution of MI can determine whether the strength of the relationship is "predictively significant" based on various tolerance levels (0.05, 0.10, 0.20, and 0.30 nats).

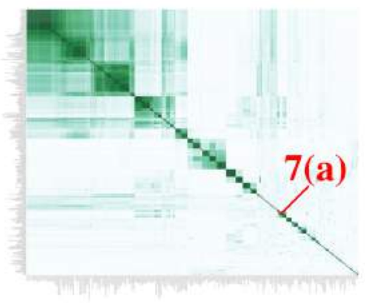
4 Applications to macroeconomic indicators of global poverty, education, and health

This section illustrates the efficacy of the proposed approach on a sparse database from an ongoing collaboration with the Bill & Melinda Gates Foundation.[1] The Gapminder data set is an extensive longitudinal dataset of ∼320 global developmental indicators for 200 countries spanning over 5 centuries [27]. These include variables from a broad set of categories such as education, health, trade, poverty, population growth, and mortality rates. We experiment with a cross-sectional slice of the data from 2002. Figure 6a shows the pairwise R^2 correlation values between all variables; each row and column in the heatmap is an indicator in the dataset, and the color of a cell is the raw value of R^2 (between 0 and 1). Figure 6b shows pairwise binary hypothesis tests of independence using HSIC [13], which detects a dense set of dependencies including many spurious relationships (Appendix B). For both methods, statistically insignificant relationships ($\alpha = 0.05$ with Bonferroni correction for multiple testing) are shown as 0. Figure 6c shows an upper bound on the pairwise probability that the MI of two variables exceeds zero (also a value between 0 and 1). These entries are estimated using Eq (7) (bypassing Monte Carlo estimation) using H=100 samples of CrossCat. Note that the metric $\mathbb{P}[\mathcal{I}_{\mathcal{G}}(x_i{:}x_j) > 0]$ in Figure 6c only indicates the *existence* of a predictive relationship between x_i and x_j; it does not quantify either the strength or directionality of the relationship.

[1]A further application, to a real-world dataset of mathematics exam scores, is shown in Appendix C.


(a) R^2 Correlation Value

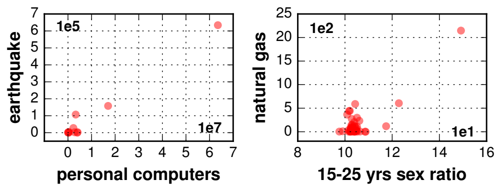

(b) HSIC [13] Independence Test



(c) $\mathbb{P}[\mathcal{I}_\mathcal{G}(x_i{:}x_j) > 0]$, Eq (7)

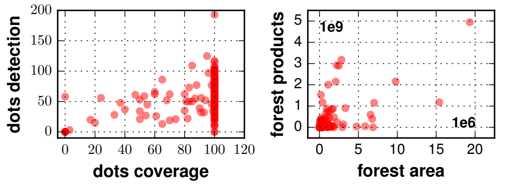
(d) Pairwise heatmaps of all 320 variables in the Gapminder dataset using three dependency detection techniques. Darker cells indicate a detected dependence between the two variables.

variable A	variable B	$\mathbb{P}[\mathcal{I}_\mathcal{G} > 0]$	R^2 $(p \ll 10^{-6})$
personal computers	earthquake affected	0.015625	0.974445
road traffic total deaths	people living w/ hiv	0.265625	0.858260
natural gas reserves	15-25 yrs sex ratio	0.031250	0.951471
forest products per ha	earthquake killed	0.046875	0.936342
flood affected	population 40-59 yrs	0.140625	0.882729



(e) Spurious relationships which are correlated under R^2, but probably independent according to posterior MI.

variable A	variable B	$\mathbb{P}[\mathcal{I}_\mathcal{G} > 0]$	R^2 $(p \ll 10^{-6})$
inflation	trade balance (% gdp)	0.859375	0.114470
DOTS detection rate	DOTS coverage	0.812500	0.218241
forest area (sq. km)	forest products (usd)	0.828125	0.206145
long term unemp. rate	total 15-24 unemp.	0.968750	NaN
male family workers	female self-employed	0.921875	NaN



(f) Common-sense relationships probably dependent according to posterior MI, but weakly correlated under R^2.

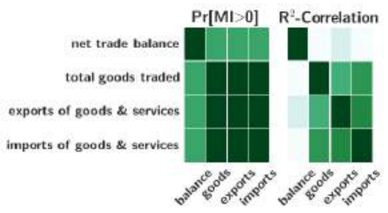
Figure 6: Comparing dependences between variables in the Gapminder dataset, as detected by R^2, HSIC (with Bonferroni correction for multiple testing), and posterior distribution over mutual information in CrossCat.

It is instructive to compare the dependencies detected by R^2 and CrossCat-Cmi. Table 6e shows pairs of variables that are spuriously reported as dependent according to correlation; scatter plots reveal they are either (i) are sparsely observed or (ii) exhibit high correlation due to large outliers. Table 6f shows common-sense relationships between pairs of variables that CrossCat-CMI detects but R^2 does not; scatter plots reveal they are either (i) non-linearly related, (ii) roughly linear with heteroskedastic noise, or (iii) pairwise independent but dependent given a third variable. Recall that CrossCat is a product of DPMMs; practically meaningful conditions for weak and strong consistency of Dirichlet location-scale mixtures have been established by [11, 33]. This supports the intuition that CrossCat can detect a broad class of predictive relationships that simpler parametric models miss.

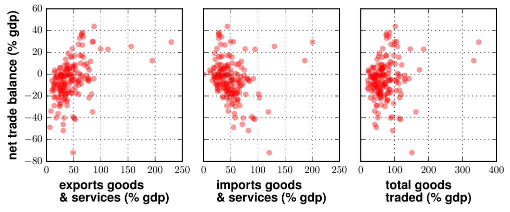
Figure 7 focuses on a group of four "trade"-related variables in the Gapminder dataset detected as probably dependent: "net trade balance", "total goods traded", "exports of goods and services", and "imports of goods and services". R^2 fails to detect a statistically significant dependence between "net trade balance" and the other variables, due to weak linear correlations and heteroskedastic noise as shown in the scatter plots (Figure 7b). From economics, these four variables are causally related by the graphical model in Figure 7c, where the value of a node is a noisy addition or subtraction of the values of its parents. Figure 7d illustrates that CrossCat recovers predictive relationships between these variables: conditioning on "exports"=150 and "balance"=30 (a low probability event according to the left subplot) centers the posterior predictive distribution of "imports" around 120, and decouples it from "total goods". The posterior CMI curves of "imports" and "total goods", with and without the conditions on "exports" and "balance", formalize this decoupling (right subplot of Figure 7d).

5 Related Work

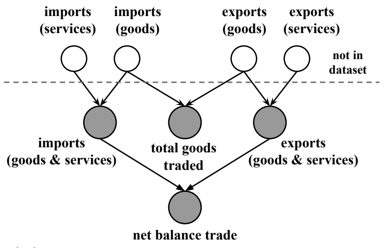
There is broad acknowledgment that new techniques for dependency detection beyond linear correlation are required. Existing approaches for conditional independence testing include the use of kernel methods [1, 10, 35, 29], copula functions [2, 25, 17], and char-



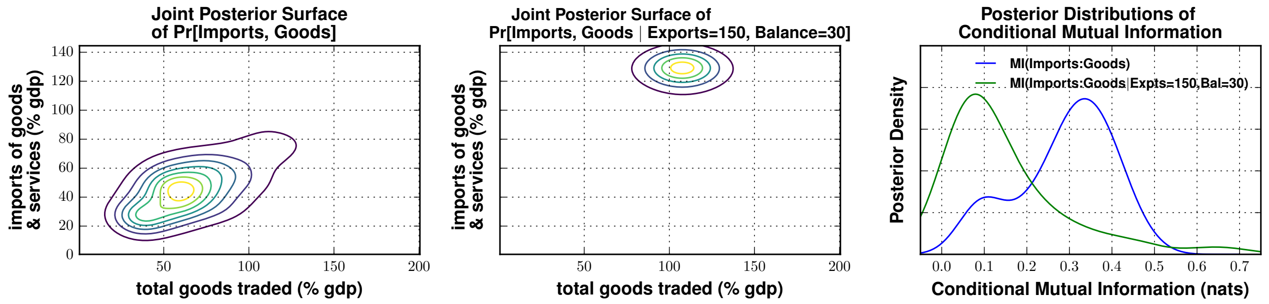
(a) Block of "trade" variables detected as probably dependent.



(b) Scatter plots show weak linear correlations and heteroskedastic noise for `net balance`.



(c) Ground truth causal structure of variables in the "trade" block.



(d) Left plot shows the joint posterior density of "imports" and "goods", where the marginal coupling is due to their common parent in (c). Center plot shows the same distribution conditioned on "exports"=150 and "balance"=30; "imports" now centers around its noiseless value of 120, and is decoupled from "goods". Right plot shows the CMI for these distributions.

Figure 7: CMI discovers existence and confirms strength of predictive relationships between "trade" variables.

acteristic functions [32], many of which capture nonlinear and multivariate predictive relationships. Unlike these methods, however, our approach represents dependence in terms of conditional mutual information and is not embedded in a frequentist decision- theoretic framework. Our quantity of interest is a full posterior distribution over CMI, as opposed to a p-value to identify when the null hypothesis CMI$=0$ cannot be rejected. Dependence detection is much less studied in the Bayesian literature; [8] use a Polya tree prior to compute a Bayes Factor for the relative evidence of dependence versus independence. Their method is used only to quantify evidence for the existence, but not assess the strength, of a predictive relationship. The most similar approach to this work was proposed independently in recent work by [16], who compute a distribution over CMI by estimating the joint density using an encompassing non-parametric Bayesian prior. However, the differences are significant. First, the Monte Carlo estimator in [16] is based on resampling empirical data. However, real-world databases may be too sparse for resampling data to yield good estimates, especially for queries given unlikely constraints. Instead, we use a Monte Carlo estimator by simulating the predictive distribution. Second, the prior in [16] is a standard Dirichlet process mixture model, whereas this paper proposes a sparsity-inducing CrossCat prior, which permits optimized computations for upper bounds of posterior probabilities as well as simplifying CMI queries with multivariate conditions.

6 Discussion

This paper has shown it is possible to detect predictive relationships by integrating probabilistic programming, information theory, and non-parametric Bayes. Users specify a broad class of conditional mutual information queries using a simple SQL-like language, which are answered using a scalable pipeline based on approximate Bayesian inference. The underlying approach applies to arbitrary generative population models, including parametric models and other classes of probabilistic programs [28]; this work has focused on exploiting the sparsity of CrossCat model structures to improve scalability for exploratory analysis. With this foundation, one may extend the technique to domains such as causal structure learning. The CMI estimator can be used as a conditional-independence test in a structure discovery algorithm such as PC [31]. It is also possible to use learned CMI probabilities as part of a prior over directed acyclic graphs in the Bayesian setting. This paper has focused on detection and preliminary assessment of predictive relationships; confirmatory analysis and descriptive summarization are left for future work, and will require an assessment of the robustness of joint density estimation when random sampling assumptions are violated. Moreover, new algorithmic insights will be needed to scale the technique to efficiently detect pairwise dependencies in very high-dimensional databases with tens of thousands of variables.

Acknowledgements

This research was supported by DARPA (PPAML program, contract number FA8750-14-2-0004), IARPA (under research contract 2015-15061000003), the Office of Naval Research (under research contract N000141310333), the Army Research Office (under agreement number W911NF-13-1-0212), and gifts from Analog Devices and Google.

References

[1] Francis Bach and Michael Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.

[2] Taoufik Bouezmarni, Jeroen VK Rombouts, and Abderrahim Taamouti. Nonparametric copula-based test for conditional independence with applications to granger causality. *Journal of Business & Economic Statistics*, 30(2):275–287, 2012.

[3] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, pages 115–123. Morgan Kaufmann Publishers Inc., 1996.

[4] National Reseach Council. *Frontiers in massive data analysis.* The National Academies Press, 2013.

[5] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications and Signal Processing. Wiley, 2012.

[6] David Edwards. *Introduction to graphical modelling.* Springer Texts in Statistics. Springer, 2012.

[7] Michael Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430): 577–588, 1995.

[8] Sarah Filippi and Chris Holmes. A bayesian nonparametric approach to testing for dependence between random variables. *Bayesian Analysis*, 2016. Advance publication.

[9] Daniel Fink. A compendium of conjugate priors. Technical report, Environmental Statistics Group, Department of Biology, Montana State University, 1997.

[10] Kenji Fukumizu, Arthur Gretton, Xiaohai Sun, and Bernhard Schölkopf. Kernel measures of conditional dependence. In *Proceedings of the Twentieth International Conference on Neural Information Processing Systems*, pages 489–496. Curran Associates Inc., 2007.

[11] Subhashis Ghosal, Jayanta Ghosh, and R.V. Ramamoorthi. Posterior consistency of dirichlet mixtures in density estimation. *The Annals of Statistics*, 27(1):143–158, 1999.

[12] Dilan Görür and Carl Edward Rasmussen. Dirichlet process gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25(4):653–664, 2010.

[13] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Proceedings of the Sixteenth International Conference Algorithmic Learning Theory*, pages 63–77. Springer, 2005.

[14] Sonia Jain and Radford M Neal. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13(1):158–182, 2012.

[15] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, 2004.

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

[17] David Lopez-Paz, Philipp Hennig, and Bernhard Schölkopf. The randomized dependence coefficient. In *Proceedings of the Twenty-Sixth International Conference on Neural Information Processing Systems*, pages 1–9. Curran Associates Inc., 2013.

[18] Vikash Mansinghka, Richard Tibbetts, Jay Baxter, Pat Shafto, and Baxter Eaves. BayesDB: A probabilistic programming system for querying the probable implications of data. *CoRR*, abs/1512.05006, 2015.

[19] Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua B. Tenenbaum. CrossCat: A fully Bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *Journal of Machine Learning Research*, 17(138):1–49, 2016.

[20] Kantilal Varichand Mardia, John T Kent, and John M Bibby. *Multivariate analysis.* Probability and Mathematical Statistics. Academic Press, 1980.

[21] Rudy Moddemeijer. On estimation of entropy and mutual information of continuous distributions. *Signal Processing*, 16(3):233–248, 1989.

[22] Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. Estimation of mutual information using kernel density estimators. *Physical Review E*, 52(3): 2318, 1995.

[23] Fritz Obermeyer, Jonathan Glidden, and Eric Jonas. Scaling nonparametric Bayesian inference via subsample-annealing. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 696–705. JMLR.org, 2014.

[24] Liam Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6):1191–1253, 2003.

[25] Barnabás Póczos, Zoubin Ghahramani, and Jeff Schneider. Copula-based kernel dependency measures. *CoRR*, abs/1206.4682, 2012.

[26] Christian Robert and George Casella. *Monte Carlo Statistical Methods.* Springer Texts in Statistics. Springer, 2005.

[27] Hans Rosling. Gapminder: Unveiling the beauty of statistics for a fact based world view. URL `https://www.gapminder.org/data/`.

[28] Feras Saad and Vikash Mansinghka. Probabilistic data analysis with probabilistic programming. *CoRR*, abs/1608.05347, 2016.

[29] Dino Sejdinovic, Arthur Gretton, and Wicher Bergsma. A kernel test for three-variable interactions. In *Proceedings of the Twenty-Sixth International Conference on Neural Information Processing Systems*, pages 1124–1132. Curran Associates Inc., 2013.

[30] Ross D Shachter. Bayes-ball: Rational pastime for determining irrelevance and requisite information in belief networks and influence diagrams. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 480–487. Morgan Kaufmann Publishers Inc., 1998.

[31] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search.* Adaptive Computation and Machine Learning. MIT Press, 2000.

[32] Liangjun Su and Halbert White. A consistent characteristic function-based test for conditional independence. *Journal of Econometrics*, 141(2): 807–834, 2007.

[33] Surya T Tokdar. Posterior consistency of dirichlet location-scale mixture of normals in density estimation and regression. *The Indian Journal of Statistics*, 68(1):90–110, 2006.

[34] Joe Whittaker. *Graphical models in applied multivariate statistics.* Wiley Series in Probability and Mathematical Statistics. Wiley, 1990.

[35] Kun Zhang, Jonas Peters, and Dominik Janzing. Kernel-based conditional independence test and application in causal discovery. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 804–813. AUAI Press, 2011.

Probabilistic Data Analysis with Probabilistic Programming

Feras Saad FSAAD@MIT.EDU
Computer Science & Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Vikash Mansinghka VKM@MIT.EDU
Department of Brain & Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Abstract

Probabilistic techniques are central to data analysis, but different approaches can be difficult to apply, combine, and compare. This paper introduces composable generative population models (CGPMs), a computational abstraction that extends directed graphical models and can be used to describe and compose a broad class of probabilistic data analysis techniques. Examples include hierarchical Bayesian models, multivariate kernel methods, discriminative machine learning, clustering algorithms, dimensionality reduction, and arbitrary probabilistic programs. We also demonstrate the integration of CGPMs into BayesDB, a probabilistic programming platform that can express data analysis tasks using a modeling language and a structured query language. The practical value is illustrated in two ways. First, CGPMs are used in an analysis that identifies satellite data records which probably violate Kepler's Third Law, by composing causal probabilistic programs with non-parametric Bayes in under 50 lines of probabilistic code. Second, for several representative data analysis tasks, we report on lines of code and accuracy measurements of various CGPMs, plus comparisons with standard baseline solutions from Python and MATLAB libraries.

Keywords: probabilistic programming, non-parametric Bayesian inference, probabilistic databases, hybrid modeling, multivariate statistics

Acknowledgments

The authors thank Taylor Campbell, Gregory Marton, and Alexey Radul for engineering support, and Anthony Lu, Richard Tibbetts, and Marco Cusumano-Towner for helpful contributions, feedback and discussions. This research was supported by DARPA (PPAML program, contract number FA8750-14-2-0004), IARPA (under research contract 2015-15061000003), the Office of Naval Research (under research contract N000141310333), the Army Research Office (under agreement number W911NF-13-1-0212), and gifts from Analog Devices and Google.

1. Introduction

Probabilistic techniques are central to data analysis, but can be difficult to apply, combine, and compare. Families of approaches such as parametric statistical modeling, machine learning and probabilistic programming are each associated with different formalisms and assumptions. This paper shows how to address these challenges by defining a new family of probabilistic models and integrating them into BayesDB, a probabilistic programming platform for data analysis. It also gives empirical illustrations of the efficacy of the framework on multiple synthetic and real-world tasks in probabilistic data analysis.

This paper introduces composable generative population models (CGPMs), a computational formalism that extends graphical models for use with probabilistic programming. CGPMs specify a table of observable random variables with a finite number of columns and a countably infinite number of rows. They support complex intra-row dependencies among the observables, as well as inter-row dependencies among a field of latent variables. CGPMs are described by a computational interface for generating samples and evaluating densities for random variables, including the (random) entries in the table as well as a broad class of random variables derived from these via conditioning. We show how to implement CGPMs for several model families such as the outputs of standard discriminative learning methods, kernel density estimators, nearest neighbors, non-parametric Bayesian methods, and arbitrary probabilistic programs. We also describe algorithms and new syntaxes in the probabilistic Metamodeling Language for building compositions of CGPMs that can interoperate with BayesDB.

The practical value is illustrated in two ways. First, the paper outlines a collection of data analysis tasks with CGPMs on a high-dimensional, real-world dataset with heterogeneous types and sparse observations. The BayesDB script builds models which combine non-parametric Bayes, principal component analysis, random forest classification, ordinary least squares, and a causal probabilistic program that implements a stochastic variant of Kepler's Third Law. Second, we illustrate coverage and conciseness of the CGPM abstraction by quantifying the lines of code and accuracy achieved on several representative data analysis tasks. Estimates are given for models expressed as CGPMs in BayesDB, as well as for baseline methods implemented in Python and MATLAB. Savings in lines of code of ~10x at no cost or improvement in accuracy are typical.

The remainder of the paper is structured as follows. Section 2 reviews related work to CGPMs in both graphical statistics and probabilistic programming. Section 3 describes the conceptual, computational, and statistical formalism for CGPMs. Section 4 formulates a wide range of probabilistic models as CGPMs, and provides both algorithmic implementations of the interface as well as examples of their invocations through the Metamodeling Language and Bayesian Query Language. Section 5 outlines an architecture of BayesDB for use with CGPMs. We show how CGPMs can be composed to form a generalized directed acyclic graph, constructing hybrid models from simpler primitives. We also present new syntaxes in MML and BQL for building and querying CGPMs in BayesDB. Section 6 applies CGPMs to several probabilistic data analysis tasks in a complex real-world dataset, and reports on lines of code and accuracy measurements. Section 7 concludes with a discussion and directions for future work.

2. Related Work

Directed graphical models from statistics provide a compact, general-purpose modeling language to describe both the factorization structure and conditional distributions of a high-dimensional joint distribution (Koller et al., 2007). Each node is a random variable which is conditionally independent of its non-descendants given its parents, and its conditional distribution given all its parents is specified by a conditional probability table or density (Nielsen and Jensen, 2009, Sec 2.3). CGPMs extend this mathematical description to a computational one, where nodes are not only random variables with conditional densities but also computational units (CGPMs) with an interface that allows them to be composed directly as software. A CGPM node typically encapsulates a more complex statistical object than a single variable in a graphical model. Each node has a set of required input variables and output variables, and all variables are associated with statistical data types. Nodes are required to both simulate and evaluate the density of a subset of their outputs by conditioning on all their inputs, as well as either conditioning or marginalizing over another subset of their outputs. Internally, the joint distribution of output variables for a single CGPM node can itself be specified by a general model which is either directed or undirected.

CGPMs combine ideas from the vast literature on modeling and inference in graphical models with ideas from probabilistic programming. This paper illustrates CGPMs by integrating them into BayesDB (Mansinghka et al., 2015a), a probabilistic programming platform for data analysis. BayesDB demonstrated that the Bayesian Query Language (BQL) can express several tasks from multivariate statistics and probabilistic machine learning in a model-independent way. However this idea was illustrated by emphasizing that a domain-general baseline model builder based on CrossCat (Mansinghka et al., 2015b), with limited support for plug-in models called “foreign predictors”, provides good enough performance for common statistical tasks. Due to limitations in the underlying formalism of generative population models (GPMs), which do not accept inputs and only learn joint distributions over observable variables, the paper did not provide an expressive modeling language for constructing a wide class of models applicable to different data analysis tasks, or for integrating domain-specific models built by experts into BayesDB. By both accepting input variables and exposing latent variables as queryable outputs, CGPMs provide a concrete proposal for mediating between automated and custom modeling using the Metamodeling Language, and model-independent querying using the Bayesian Query Language. The CGPM abstraction thus exposes the generality of BQL to a much broader model class than originally presented, which includes hybrids models with generative and discriminative components.

It is helpful to contrast CGPMs in BayesDB with other probabilistic programming formalisms such as Stan (Carpenter et al., 2015). Stan is a probabilistic programming language for specifying hierarchical Bayesian models, with built-in algorithms for automated, highly efficient posterior inference. However, it is not straightforward to (i) integrate models from different formalisms such as discriminative machine learning as sub-parts of the overall model, (ii) directly query the outputs of the model for downstream data analysis tasks, which needs to be done on a per-program basis, and (iii) build composite programs out of smaller Stan programs, since each program is an independent unit without an interface. CGPMs provide an interface for addressing these limitations and makes it possible to wrap Stan programs as CGPMs that can then interact, through BayesDB, with CGPMs implemented in other systems.

Tabular (Gordon et al., 2014) is a schema-driven probabilistic programming language which shares some similarity to composable generative population models. For instance, both the statistical representation of a CGPM (Section 3.3), and a probabilistic schema in Tabular, characterize a data generating process in terms of input variables, output variables, latent variables, parameters and hyper-parameters. However, unlike Tabular schemas, CGPMs explicitly provide a computational interface, which is more general than the description of their internal structure, and facilitates their composition (Section 5.2). In Tabular, probabilistic programs are centered around parametric statistical modeling in factor graphs, where the user manually constructs variable nodes, factor nodes, and the quantitative relationships between them. On the other hand, CGPMs express a broad range of model classes which do not necessarily naturally admit natural representations as factor graphs, and combine higher-level automatic model discovery (using baseline generative CGPMs) with user-specified overrides for hybrid modeling.

3. Composable Generative Population Models

In this section we describe composable generative population models (CGPMs), a computational abstraction that provides a uniform treatment of a broad class of models and methods in probabilistic data analysis. This section is divided into three parts. The first part formalizes the notion of a statistical population in terms of a random tabular data structure with a finite number of columns and a countably infinite number of rows, and establishes notation used throughout the paper. The second part outlines the computational interface that defines CGPMs. The third part describes a class of statistical graphical models which can be naturally expressed using the CGPM framework.

3.1 Populations

In our framework, a population $\mathcal{P}$ is defined in terms of a finite set of variables $(v_1, \dots, v_T)$, where each variable v_t takes values in a general observation space $\mathcal{V}_t$. Each variable has a qualitative interpretation as a particular property or attribute of the members of the population. The r^{th} member of the population, denoted $\boldsymbol{x}_r$, is a T-dimensional vector $(x_{[r,1]}, \dots, x_{[r,T]})$, and the element $x_{[r,t]}$ is a variable corresponding to the variable v_t of member r. The entire population is then organized as an infinite exchangeable sequence $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots)$ of members.

The population can be conceptualized as a tabular data structure with a finite number of columns and an infinite number of rows. Column t corresponds to variable v_t, row r to member $\boldsymbol{x}_r$, and cell (r, t) to element $x_{[r,t]}$. The table is further associated with the observation spaces $\{\mathcal{V}_t : t \in [T]\}$. The exchangeability assumption translates into the requirement that $\mathcal{P}$ is unchanged by permuting the member ids. Finally, a measurement is defined as an observed value for cell (r, t) in the data structure. In general, we use $x_{[r,t]}$ to indicate the element as a variable as well as its measured value (if one exists); the meaning is typically clear from context. A collection of measurements recorded in the infinite table is referred to as a dataset $\mathcal{D}$.

It is helpful to compare the standard notion of a statistical population with the formalism described above. In classical multivariate statistics, a data analysis tasks starts with a "data matrix", a finite array containing the measurements from some experiment, and additional modeling assumptions then specify that these measurements are a "random sample" from a statistical population. The members of the population are generated by a distribution (often a multivariate normal) whose unknown parameters (population mean, population covariance, etc) we wish to discover (Timm, 2002; Khattree and Naik, 2000; Gelman and Hill, 2006). This usage of the term "statistical population" thus combines domain knowledge (in defining the schema), observed data, and quantitative modeling assumptions (in terms of the random variables) under one umbrella idea.

By contrast, our framing characterizes a population only in terms of a set of population variables and their observation spaces. This framing does not commit to a probabilistic description of the data generating process, and is intended to invite questions about populations without reference to an underlying statistical model. Moreover, every member in our definition of a population is associated with a unique identifier – while this paper only focuses on modeling measurements conditioned on the member ids, in principle the member ids themselves could be modeled by a process that is more complex than random sampling.

Moreover, our mathematical specification of a population attempts to be more granular than the standard formalism from multivariate statistics. We explicitly differentiate between a variable v_t, and the set of elements $\{x_{[r,t]} : r = 1, 2, \dots\}$ which are versions of that variable v_t for each member. By separating a variable (a "column" in the infinite table) from its related element-level variables ("cells" in that column), and carefully accounting for all elements in the data structure, we can discuss precisely the mathematical and algorithmic operations performed by CGPMs. This level of analysis would not be possible had we coarsely specified a population as a single random vector $\boldsymbol{x} = (x_1, \dots, x_T)$, and viewed measurements collected in a "data matrix" as independent realizations of $\boldsymbol{x}$. Moreover, specifying measurements at the cell level deals with arbitrary/sparse patterns of observations in the infinite table, in contrast with the standard notion of data matrices which are often treated as objects from linear algebra. Similarly, explicitly notating the observation spaces $\{\mathcal{V}_t : t \in [T]\}$ allows us to capture heterogeneity in population variables, rather than assume the universe is T-dimensional Euclidean space. These characteristics are common in real-world populations that arise in probabilistic data analysis.

3.2 Computational description of composable generative population models

Having established populations, we now introduce composable generative population models in terms of the computational interface they provide. A composable generative population model (CGPM) $\mathcal{G}$ characterizes the data generating process for a population $\mathcal{P}$. The CGPM selects from the population variables $(v_1, v_2, \dots, v_T)$ a set of output variables $(v_1^{out}, \dots, v_O^{out})$ and a set of input variables $(v_1^{in}, \dots, v_I^{in})$. For each member r, $\mathcal{G}$ is responsible for modeling the full joint distribution of all the output variables conditioned on all the input variables. CGPMs differ from the mathematical definition of a probability density in that they are defined directly in terms of a computational interface, as shown in Listing 1. This interface explicitly differentiates between the *sampler* of a random variable from its conditional distribution, and the *assessor* of its conditional density.

Listing 1 Computational interface for composable generative population models.

- $\mathcal{G} \leftarrow$ `create`(`population`: $\mathcal{P}$, `outputs`: $\{v_i^{out}\}_{i\in[O]}$, `inputs`: $\{v_j^{in}\}_{j\in[I]}$, `binary`: $\mathcal{B}$, `seed`: s)

  Create a CGPM for the population, with the specified inputs and outputs.

- $\mathbf{s} \leftarrow$ `simulate` ($\mathcal{G}$, `member`: r, `query`: $Q = \{q_k\}$, `evidence` : $E = \{x_{[r,e_j]}\} \cup \boldsymbol{y}_r$)

  Generate a sample from the distribution $\mathbf{s} \sim^{\mathcal{G}} \boldsymbol{x}_{[r,Q]} | \{\boldsymbol{x}_{[r,E]}, \boldsymbol{y}_r, \mathcal{D}\}$.

- $c \leftarrow$ `logpdf` ($\mathcal{G}$, `member`: r, `query` : $Q = \{x_{[r,q_k]}\}$, `evidence` : $E = \{x_{[r,e_j]}\} \cup \boldsymbol{y}_r$)

  Evaluate the log density $\log p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]} | \{\boldsymbol{x}_{[r,E]}, \boldsymbol{y}_r, \mathcal{D}\})$.

- $\mathcal{G}' \leftarrow$ `incorporate` ($\mathcal{G}$, `measurement` : $x_{[r,k]}$)

  Record a measurement $x_{[r,k]} \in \mathcal{V}_k$ into the dataset $\mathcal{D}$.

- $\mathcal{G}' \leftarrow$ `unincorporate` ($\mathcal{G}$, `member` : r)

  Eliminate all measurements of input and output variables for member r.

- $\mathcal{G}' \leftarrow$ `infer` ($\mathcal{G}$, `program` : $\mathcal{T}$)

  Adjust internal state in accordance with the learning procedure specified by program $\mathcal{T}$.

There are several key ideas to draw from the interface. In `create`, $\mathcal{P}$ contains the set of all population variables and their observation spaces. The `binary` is an opaque probabilistic program containing implementations of the interface, and `seed` is the entropy source from which the CGPM draws random bits. The `outputs` requires at least one entry, the `inputs` may be an empty set, and any variable which is neither an input nor an output is unmodeled by the CGPM. For simplicity, we use the symbol $x_{[r,t]}$ to denote the output variable $x_{[r,v_t^{out}]}$ and similarly $y_{[r,t]}$ for input variable $y_{[r,v_t^{in}]}$ of member r. These elements are often collected into vectors $\boldsymbol{x}_r$ and $\boldsymbol{y}_r$, respectively

In `incorporate`, measurements are recorded at the cell-level, allowing only a sparse subset of observations for member r to exist. The `measurement` may be either an output element from $\boldsymbol{x}_r$ or input element from $\boldsymbol{y}_r$.

Both `simulate` and `logpdf` are computed for single member r of the population. The `query` parameter differs between the two methods: in `simulate`, $Q = \{q_k\}$ is a set of indices of output variables that are to be simulated jointly; in `logpdf`, $Q = \{x_{[r,q_k]}\}$ is a set of values for the output variables whose density is to be assessed jointly. The `evidence` parameter is the same for both `simulate` and `logpdf`, which contains additional information about r, possibly including the values of a set of output variables that are disjoint from the query variables. In particular, if $x_{[r,E]}$ is empty, the CGPM is asked to marginalize over all its output variables that are not in the query Q; if $x_{[r,E]}$ is not empty, the CGPM is required to condition on those output values.

The target distributions in `simulate` and `logpdf` are also conditioned on all previously incorporated measurements in the dataset $\mathcal{D}$. Because CGPMs generally model populations with inter-row dependencies, measurements of other members $s \neq r$ are relevant to a `simulate` or `logpdf` query about r. The CGPM interface allows the user to override a previous measurement of r in $\mathcal{D}$ on a per-query basis; this occurs when an element $x_{[r,e_j]}$ or $\boldsymbol{y}_r$ in the `evidence` contradicts an existing measurement $x'_{[r,e_j]}$ or $\boldsymbol{y}'_r$ in $\mathcal{D}$. Asking such hypothetical queries addresses several tasks of interest in probabilistic data analysis, such as simulating "what-if" scenarios and detecting outliers in high-dimensional populations.

Finally, the `infer` procedure evolves the CGPM's internal state in response to the inflow of measurements. The inference program $\mathcal{T}$ can be based on any learning strategy applicable to the CGPM, such as Markov Chain Monte Carlo transitions, variational inference, maximum-likelihood, least-squares estimation, or no learning.

3.3 Statistical description of composable generative population models

The previous section outlined the external interface that defines a CGPM without specifying its internal structure. In practice, many CGPMs can be described using a general graphical model with both directed and undirected edges. The data generating process is characterized by a collection of variables in the graph,

$$\mathcal{G} = (\boldsymbol{\alpha}, \boldsymbol{\theta}, \boldsymbol{Z} = \{\boldsymbol{z}_r\}_{r=1}^{\infty}, \boldsymbol{X} = \{\boldsymbol{x}_r\}_{r=1}^{\infty}, \boldsymbol{Y} = \{\boldsymbol{y}_r\}_{r=1}^{\infty}).$$

- $\boldsymbol{\alpha}$: Fixed quantities such as input and output dimensionalities, observation spaces, dependence structures and statistical hyperparameters.

- $\boldsymbol{\theta}$: Population-level, or global, latent variables relevant to all members.

- $\boldsymbol{z}_r = (z_{[r,1]}, \dots, z_{[r,L]})$: Member-specific latent variables governing only member r directly. A subset of these variables may be exposed, and treated as queryable output variables.

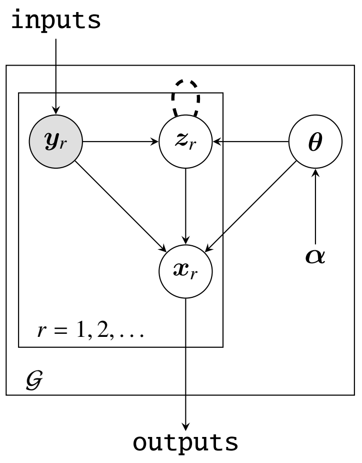


Figure 1: Internal independence constraints for a broad class of composable generative population models. All nodes in the diagram are multidimensional. Internally, the hyperparameters $\boldsymbol{\alpha}$ are fixed and known quantities. The global latents $\boldsymbol{\theta}$ are shared by all members of the population. Member-specific latents $\boldsymbol{z}_r$ interact only with their corresponding observations $\boldsymbol{x}_r$, as well as other member-latents $\{\boldsymbol{z}_s : s \neq r\}$ as indicated by the dashed loop around the plate. Nodes $\boldsymbol{x}_r$ and $\boldsymbol{x}_s$ across different members r and s are independent conditioned on their member-latents. However, general dependencies are permitted within elements $\{x_{[r,i]} : i \in [O]\}$ of node $\boldsymbol{x}_r$. The input variables $\boldsymbol{y}_r$ are ambient conditioning variables in the population and are always observed; in general, $\boldsymbol{y}_r$ may be the output of another CGPM (Section 5.2). Externally, $\mathcal{G}$ is specified by an opaque `binary`, e.g. a probabilistic program, describing the data generating process, and `outputs` and `inputs` that specify the variable names for `simulate` and `logpdf`.

- $\boldsymbol{x}_r = (x_{[r,1]}, \ldots, x_{[r,O]})$: Output variables representing observable attributes of member r.
- $\boldsymbol{y}_r = (y_{[r,1]}, \ldots y_{[r,I]})$: Input variables that must be present for any query about $\boldsymbol{x}_r$, such as the "feature vectors" in a discriminative model.

The notion of global and local latent variables is a common motif in the hierarchical modeling literature (Blei et al., 2016). They are useful in specifying the set of constraints governing the dependence between observable variables in terms of some latent structure. From this lens, CGPMs satisfy the following conditional independence constraint,

$$\forall r \neq s \in \mathbb{N}, \forall j, k \in [O] : x_{[r,j]} \perp\!\!\!\perp x_{[s,k]} \mid \{\boldsymbol{\alpha}, \boldsymbol{\theta}, \boldsymbol{z}_r, \boldsymbol{z}_s\}. \tag{1}$$

Equation (1) formalizes the notion that all dependencies across members $r \in \mathbb{N}$ are fully mediated by the global parameters $\boldsymbol{\theta}$ and member-specific variables $\{\boldsymbol{z}_r\}$. However, elements $x_{[r,j]}$ and $x_{[r,i]}$ within a member are free to assume any dependence structure, allowing for arbitrary inter-row dependencies. This feature allows CGPMs to express undirected models where the output variables are not exchangeably-coupled, such as Gaussian Markov random fields (Rue and Held, 2005).

A common specialization of constraint (1) further requires that the member-specific latent variables $\{\boldsymbol{z}_r\}$ are conditionally independent given $\boldsymbol{\theta}$; a comprehensive list of models in machine learning and statistics satisfying this additional constraint is given in (Hoffman et al., 2013, Section

2.1). However, CGPMs permit more general dependencies in that member latents may be coupled conditioned $\boldsymbol{\theta}$, thus allowing for complex intra-row dependencies. CGPMs can thus be used for models such as Gaussian process regression with noisy observations (Rasmussen and Williams, 2006), where the member-specific latent variables (i.e. the noiseless observations) across different members in the population are jointly Gaussian (Damianou and Lawrence, 2013, Figure 1).

Figure 1 summarizes these ideas by showing a CGPM as a graphical model. Finally, we note it is also possible for a CGPM to fully implement the interface without admitting a "natural" representation in terms of the graphical structure from Figure 1, as shown by several examples in Section 4.

3.4 Composable generative population models are an abstraction for probabilistic processes

By providing a computational interface, the CGPM interface provides a layer of abstraction which separates the internal implementation of a probabilistic model from the generative process it represents. In this section we will explore how the computational (external) description of a CGPM provides a fundamentally different understanding than its statistical (internal) description.

As an example, consider a Dirichlet process mixture model (Antoniak, 1974) expressed as a CGPM. The hyperparameters $\boldsymbol{\alpha} = (H, \gamma, F)$ are the base measure H, concentration parameter γ, and parametric distribution F of the observable variables $\{\boldsymbol{x}_r\}$. The member latent variable $\boldsymbol{z}_r = (z_r)$ is the cluster assignment of r. Consider now two different representations of the underlying DP, each leading to a different notion of (i) population parameters $\boldsymbol{\theta}$, and (ii) conditional independence constraints.

- In the stick breaking representation (Sethuraman, 1994), the population parameters $\boldsymbol{\theta} = \{(\phi_i, \pi_i) : i \in \mathbb{N}\}$, where ϕ_i are the atoms that parameterize the likelihood $F(\cdot|\phi_i)$ (drawn i.i.d from H) and π_i their weights (drawn jointly from GEM(γ)). Conditioned on $\{\boldsymbol{\alpha}, \boldsymbol{\theta}\}$, the member latents are independent, $z_r \sim^{iid} \textsc{Categorical}(\{\pi_1, \pi_2, \ldots\})$.
- In the Chinese restaurant process representation (Aldous, 1985), the population parameters $\boldsymbol{\theta} = \{\phi_i : i \in \mathbb{N}\}$ are now only the atoms, and the weights are fully collapsed out. Conditioned on $\{\boldsymbol{\alpha}, \boldsymbol{\theta}\}$, the member latents are exchangeably coupled $\{z_1, z_2, \ldots\} \sim \textsc{Crp}(\gamma)$.

These internal representation choices are not exposed by the CGPM interface and may be interchanged without altering the queries it can answer.[1] It follows that the computational description of CGPMs provides an abstraction boundary between a particular implementation of a probabilistic model and the generative process for the population that it represents. Two implementations of a CGPM may encapsulate the same process by inducing an identical marginal distribution over their observable variables, while maintaining different auxiliary-variable representations internally.

The encapsulation of a CGPM's internal state can be relaxed by asking the CGPM to expose member-specific latent variables as outputs. In terms of the infinite table metaphor from Section 3.1, this operation may be conceptualized as the CGPM "fantasizing" the existence of new columns in the underlying population. Providing a gateway into the internal state of a CGPM trades-off the model independence of the interface with the ability to query the hidden structure of a particular probabilistic process. Section 5 describes surface-level syntaxes for exposing latent variables, and Section 6.1 illustrates its utility for inferring latent cluster assignments in an infinite mixture model, as well simulating projections of high-dimensional data onto low-dimensional latent subspaces.

1. However, it is important to note that interchanging representations may result in different performance characteristics, such as compute time or approximateness of `simulate` and `logpdf`.

4. Algorithmic Implementations of Composable Generative Population Models

In this section, we illustrate that the computational abstraction of CGPMs is applicable to broad classes of modeling approaches and philosophies. Table 1 shows the collection of models whose internal structure we will develop from the perspective of CGPMs. Section 6 shows both comparisons of these CGPMs and their practical application to data analysis tasks.

	Composable Generative Population Model	**Modeling Approach**
Section 4.2	Cross Categorization	non-parametric Bayesian generative modeling
Section 4.3	Ensemble Classifiers and Regressors	discriminative machine learning
Section 4.4	Factor Analysis & Probabilistic PCA	dimensionality reduction
Section 4.5	Parametric Mixture of Experts	discriminative statistical modeling
Section 4.7	Multivariate Kernel Density Estimation	classical multivariate statistics
Section 4.6	Generative Nearest Neighbors	clustering based generative modeling
Section 4.8	Probabilistic Programs in VentureScript	probabilistic programming

Table 1: Examples of composable generative population models, and a modeling framework for data analysis to which they belong.

The two methods from the interface in Listing 1 whose algorithmic implementations we outline for each CGPM are

- $\mathbf{s} \leftarrow$ `simulate` $(\mathcal{G}$, `member`: r, `query`: $Q = \{q_k\}$, `evidence` : $E = \{x_{[r,e_j]}\} \cup \boldsymbol{y}_r)$
  Generate a sample from the distribution $\mathbf{s} \sim^{\mathcal{G}} \boldsymbol{x}_{[r,Q]} | \{\boldsymbol{x}_{[r,E]}, \boldsymbol{y}_r, \mathcal{D}\}$.
- $c \leftarrow$ `logpdf` $(\mathcal{G}$, `member`: r, `query` : $Q = \{x_{[r,q_k]}\}$, `evidence` : $E = \{x_{[r,e_j]}\} \cup \boldsymbol{y}_r)$
  Evaluate the log density $\log p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]} | \{\boldsymbol{x}_{[r,E]}, \boldsymbol{y}_r, \mathcal{D}\})$.

In both `simulate` and `logpdf`, the target distributions for the query variables $\boldsymbol{x}_{[r,Q]}$ require an implementation of two operations:

- Conditioning on the evidence variables $\boldsymbol{x}_{[r,E]}$, in addition to the input variables $\boldsymbol{y}_r$ and entire measurement set $\mathcal{D}$.
- Marginalizing over all output variables $\{x_{[r,i]} : i \in [O] \backslash (E \cup Q)\}$ not in the query or evidence.

Both conditioning and marginalizing over joint distributions allow users of CGPMs to pose non-trivial queries about populations that arise in multivariate probabilistic data analysis. All our algorithms generally assume that the information known about member r in `simulate` and `logpdf` is only what is provided for the `evidence` parameter. Extending the implementations to deal with observed members $r' \in \mathcal{D}$ is mostly straightforward and often implementation-specific. We also note that the figures in these subsections contain excerpts of probabilistic code in the Bayesian Query Language, Metamodeling Language, and VentureScript; most of their syntaxes are outlined in Section 5. Finally, we leave the many possible implementations of `infer` for each CGPM, which learns the latent state using observed data, primarily to external references.

4.1 Primitive univariate distributions and statistical data types

The statistical data type of a population variable v_t provides a more refined taxonomy than the "observation space" $\mathcal{V}_t$ described in Section 3.1. Table 2 shows the collection of statistical data types available in the Metamodeling Language (Section 5.3), out of which more complex CGPMs are built. The (parameterized) support of a statistical type defines the set in which samples from `simulate` take values. Each statistical type is also associated with a base measure which ensures `logpdf` is well-defined. In high-dimensional populations with heterogeneous types, `logpdf` is taken against the product measure of these univariate base measures. The statistical type also identifies invariants that the variable maintains. For instance, the values of a `NOMINAL` variable are permutation-invariant; the distance between two values for a `CYCLIC` variable is defined circularly (modulo the period), etc. The final column in Table 2 shows the primitive univariate CGPMs that are compatible with each statistical type. For these simple CGPMs, `logpdf` is implemented directly from their probability density functions, and algorithms for `simulate` are well-known (Devroye, 1986). For `infer`, the CGPMs may have fixed parameters, or learn from data using i.e. maximum likelihood (Casella and Berger, 2002, Ch. 7) or Bayesian priors (Fink, 1997).

Statistical Data Type	Parameters	Support	Measure/σ-Algebra	Primitive Univariate CGPM
BINARY	-	$\{0, 1\}$	$(\#, 2^{\{0,1\}})$	BERNOULLI
NOMINAL	symbols: S	$\{0, 1, \dots, S-1\}$	$(\#, 2^{[S]})$	CATEGORICAL
COUNT/RATE	base: b	$\{0, \frac{1}{b}, \frac{2}{b}, \dots\}$	$(\#, 2^{\mathbb{N}})$	POISSON, GEOMETRIC
CYCLIC	period: p	$(0, p)$	$(\lambda, \mathcal{B}(\mathbb{R}))$	VON-MISES
MAGNITUDE	–	$(0, \infty)$	$(\lambda, \mathcal{B}(\mathbb{R}))$	LOGNORMAL, EXPONENTIAL
NUMERICAL	–	$(-\infty, \infty)$	$(\lambda, \mathcal{B}(\mathbb{R}))$	NORMAL
NUMERICAL-RANGED	low: l, high:h	$(l, h) \subset \mathbb{R}$	$(\lambda, \mathcal{B}(\mathbb{R}))$	BETA, NORMAL-TRUNC

Table 2: Statistical data types, and their supports, base measures, and primitive CGPMs.

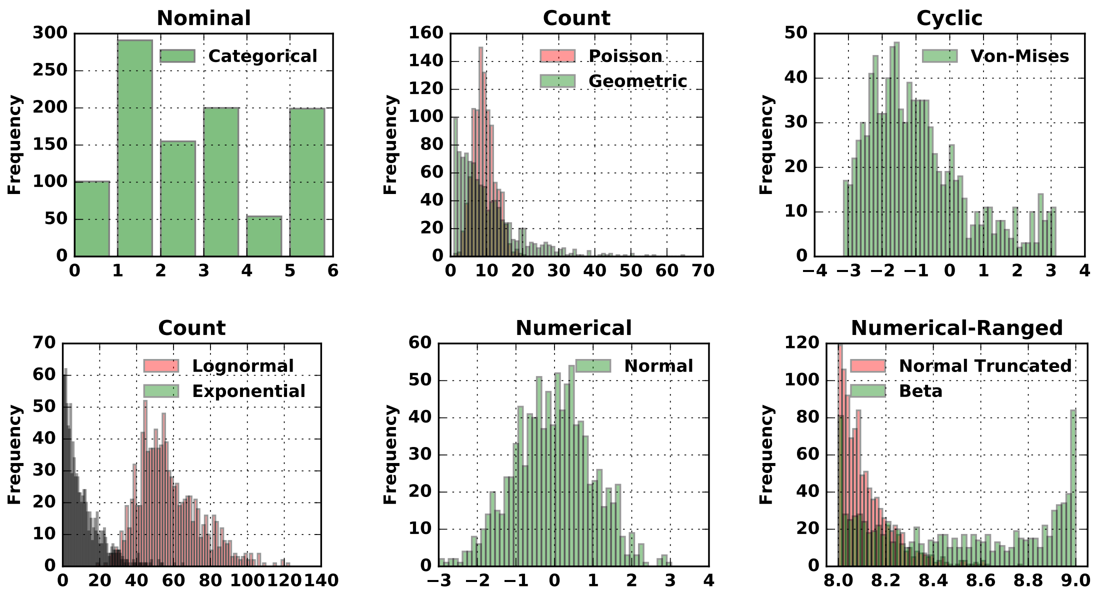


Figure 2: Samples from the primitive CGPMs of each statistical data type.

4.2 Cross-Categorization

Cross-Categorization (CrossCat) is a Bayesian non-parametric method for learning the joint distribution over all variables in a heterogeneous, high-dimensional population (Mansinghka et al., 2015b). The generative model begins by first partitioning the set of variables $(v_1, \dots, v_T)$ into blocks. This step is CrossCat's "outer" clustering, since it partitions the "columns"(when viewing the population in terms of its infinite table representation from Section 3.1). Let π denote the variable partition, and $\{B_i : i \in |\pi|\}$ denote its blocks. π is a global latent variable which dictates the structural dependencies between variables; any collection of variables in different blocks are mutually independent, and all variables in the same block are mutually dependent. It follows that for each member r, the joint distribution for $\boldsymbol{x}_r$ factorizes,

$$p_{\mathcal{G}}(\boldsymbol{x}_r|\boldsymbol{\theta}) = \prod_{B \in \pi} p_{\mathcal{G}}(\boldsymbol{x}_{[r,B]}|\boldsymbol{\theta}_B).$$

The bundle of global parameters $\boldsymbol{\theta}$ includes π as well as a set of block-specific latent variables $\{\boldsymbol{\theta}_B\}_{B\in\pi}$. Within each block B of dependent variables, the elements $\{x_{[r,i]}, i \in B\}$ are conditionally independent given a member-specific latent variable $z_{[r,B]} \in \mathbb{N}$. This variable is an "inner" clustering assignment in CrossCat, since it specifies the cluster identity of row r with respect to the variables in block B. The joint distribution over elements then factorizes,

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,B]}|\boldsymbol{\theta}_B) = \sum_k \left[\left(\prod_{i \in B} p_{\mathcal{G}}(x_{[r,i]}|\phi_{[i,k]}) \right) p_{\mathcal{G}}(z_{[r,B]} = k|\boldsymbol{\omega}_B) \right]. \tag{2}$$

The global parameter $\phi_{[i,k]}$ parameterizes the primitive univariate CGPM (of the appropriate statistical type) for v_i in cluster k, and $\boldsymbol{\omega}_B$ is a parameter governing the distribution of the latent variable $z_{[r,B]}$. This description fully specifies the CrossCat factorization of the joint distribution $p_{\mathcal{G}}(\boldsymbol{x}_r|\boldsymbol{\theta})$. This generative template is encoded into a hierarchical Bayesian model by specifying priors over the partition π, mixture weights $\boldsymbol{\omega}_B$ in each block $B \in \pi$, and distributional parameters $\phi_{[i,k]}$. In contrast to (Mansinghka et al., 2015b), Algorithm 2a presents (for simplicity) a fully uncollapsed representation of the CrossCat prior, using a GEM distribution (Pitman, 2002) for the inner DP.

Having described the generative process and established notation, we now outline algorithms for `logpdf` and `simulate`. Since CrossCat is a Bayesian CGPM, the distribution of interest $p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \mathcal{D})$ requires us to marginalize out the latent variables $(\boldsymbol{\theta}, \boldsymbol{Z})$. Sampling from the posterior is covered in (Mansinghka et al., 2015b, Section 2.4), so we only focus on implementing `simulate` and `logpdf` assuming posterior samples of latents are available.[2] These implementations are summarized in Algorithms 2b and 2c, where all routines have access to a posterior sample of the latent variables in Algorithm 2a. While our algorithms are based on an uncollapsed CrossCat, in practice, the Parameter-Prior and primitive CGPMs from lines 8 and 13 in Algorithm 2a form a conjugate pair. The density terms $p_{\mathcal{G}}(x_{[r,c]}|\phi_{[c,k]})$ are computed by marginalizing $\phi_{[c,k]}$, and using the sufficient statistics in cluster k along with the column hyperparameters $\boldsymbol{\lambda}_i$, i.e. $p_{\mathcal{G}}(x_{[r,c]}|\{x_{[r',c]} : z_{[r',B]} = k\}, \boldsymbol{\lambda}_i)$. This Rao-Blackwellization enhances the inferential quality and predictive performance of CrossCat, and the one sample approximation on line 6 of Algorithm 2d, an instance of Algorithm 8 from (Neal, 2000), becomes exact for evaluating `logpdf`. Section 3.4 contains a discussion on the implications of different internal representations of a generative process (such as collapsed or uncollapsed) from the perspective of CGPMs.

2. Section 4.8 outlines the Monte Carlo estimator for aggregating the samples in a general probabilistic programming setting.

Algorithm 2a Forward sampling a population in the CrossCat CGPM.

```
 1: $\alpha \sim$ Crp-Concentration-Prior                          ▷ sample a concentration for the outer CRP
 2: $\pi \sim$ Crp$(\alpha|[T])$                                     ▷ sample partition of variables $\{v_1, \dots, v_T\}$
 3: for $B \in \pi$ do                                                ▷ for each block $B$ in the variable partition
 4:     $\alpha_B \sim$ Crp-Concentration-Prior                      ▷ sample a concentration for the inner CRP at $B$
 5:     $(\omega_{[B,1]}, \omega_{[B,2]}, \dots) \sim$ GEM$(\alpha_B)$   ▷ sample stick-breaking weights of its clusters
 6: for $i \in [T]$ do                                                ▷ for each variable $v_i$ in the population
 7:     $\boldsymbol{\lambda}_i \sim$ Parameter-Hyper-Prior            ▷ sample hyperparams from a hyperprior
 8:     $(\phi_{[i,1]}, \phi_{[i,2]}, \dots) \overset{iid}{\sim}$ Parameter-Prior$(\boldsymbol{\lambda}_i)$   ▷ sample component distribution params
 9: for $r = 1, 2, \dots$ do                                          ▷ for each member $r$ in the population
10:     for $B \in \pi$ do                                            ▷ for each block $B$ in the variable partition
11:         $z_{[r,B]} \sim$ Categorical$(\boldsymbol{\omega}_B)$        ▷ sample the cluster assignment of $r$ in $B$
12:         for $i \in B$ do                                          ▷ for each variable $v_i$ in the block
13:             $x_{[r,i]} \sim p_{\mathcal{G}}(\cdot|\phi_{[i,z_{[r,B]}]})$   ▷ sample observable element $v_i$ for $r$
```

Algorithm 2b `simulate` for the CrossCat CGPM.

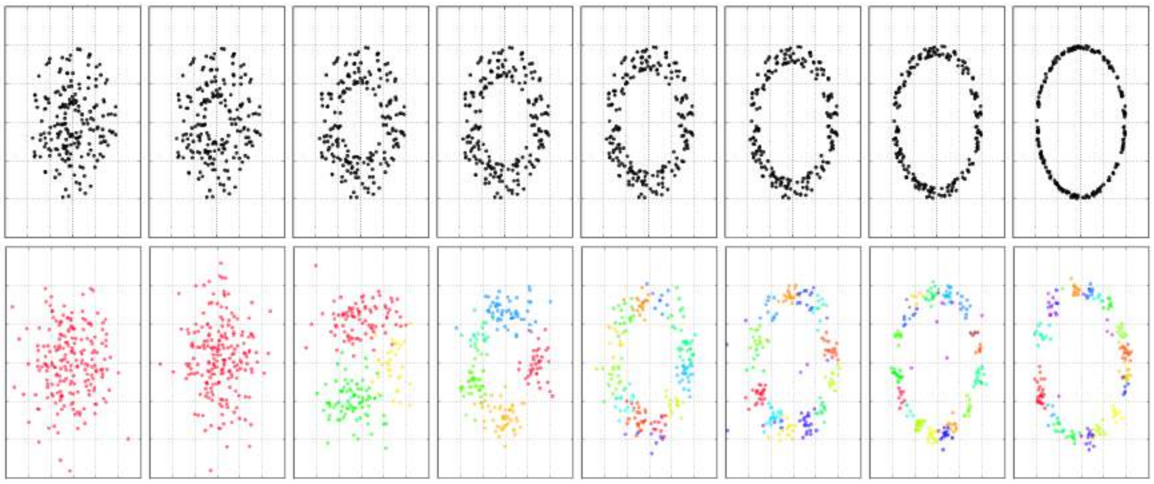
```
1: function Simulate
2:     $\boldsymbol{x}_{[r,Q]} \leftarrow \varnothing$                    ▷ initialize empty sample
3:     for $B \in \pi$ do                                            ▷ for each block $B$ in the variable partition
4:         $\boldsymbol{l} \leftarrow$ Compute-Cluster-Probabilities$(B)$   ▷ retrieve posterior probabilities of proposal clusters
5:         $z_{[r,B]} \sim$ Categorical$(\boldsymbol{l})$               ▷ sample a cluster
6:         for $q \in (Q \cap B)$ do                                  ▷ for each query variable in the block
7:             $x_{[r,q]} \sim p_{\mathcal{G}}(\cdot|\phi_{[q,z_{[r,B]}]})$   ▷ sample an observation element
8:     return $\boldsymbol{x}_{[r,Q]}$                                  ▷ overall sample of query variables
```

Algorithm 2c `logpdf` for the CrossCat CGPM.

```
1: function LogPdf
2:     for $B \in \pi$ do                                            ▷ for each block $B$ in the variable partition
3:         $\boldsymbol{l} \leftarrow$ Compute-Cluster-Probabilities$(B)$   ▷ retrieve posterior probabilities of proposal clusters
4:         $K \leftarrow |\boldsymbol{l}|$                                 ▷ compute number of proposed clusters
5:         $t_B \leftarrow \sum_{k=1}^{K} \left[ \left( \prod_{q \in (Q \cap B)} p_{\mathcal{G}}(x_{[r,q]}|\phi_{[r,k]}) \right) \frac{l_k}{\sum_{k'=1}^{K} l_{k'}} \right]$   ▷ compute density for query variables in $B$
6:     return $\sum_{B \in \pi} \log(t_B)$                               ▷ overall log density of query
```

Algorithm 2d Computing the cluster probabilities in a block of the CrossCat partition.

```
1: function Compute-Cluster-Probabilities (block: $B$)
2:     $K \leftarrow \max_{r' \in \mathcal{D}} \{z_{[r',B]}\}$             ▷ compute number of occupied clusters
3:     for $k = 1, 2, \dots, K$ do $c_k = |\{r' \in \mathcal{D} : z_{[r',B]} = k\}|$   ▷ compute number of members in each cluster
4:     for $k = 1, 2, \dots, K$ do                                    ▷ for each cluster $k$
5:         $l_k \leftarrow \left( \frac{c_k}{\sum_j c_j + \alpha_B} \right) \prod_{e \in (E \cap B)} p_{\mathcal{G}}(x_{[r,e]}|\phi_{[e,k]})$   ▷ compute probability of $r$ joining $k$
6:     $l_{K+1} \leftarrow \left( \frac{\alpha_B}{\sum_j c_j + \alpha_B} \right) \prod_{e \in (E \cap B)} p_{\mathcal{G}}(x_{[r,e]}|\phi_{[e,K+1]})$   ▷ compute probability of $r$ in singleton cluster
7:     return $(l_1, \dots, l_K, l_{K+1})$                              ▷ normalized probabilities of proposed clusters
```


(a) Black dots represent observed samples from a noisy ring with decreasing noise level. Colored dots represent samples from CrossCat's posterior predictive after two minutes of analysis. The color of a point indicates its latent cluster assignment from CrossCat's inner Dirichlet process mixture. This panel illustrates a phenomenon known as the Bayes Occam's razor. At higher noise levels (left side plots) there is less evidence for patterns in the data, so the posterior prefers a less complex model with a small number of large clusters. At lower noise levels (right side plots) there is more evidence for the functional relationship, so the posterior prefers a more complex model with a large number of small clusters, which is required to emulate the ring.

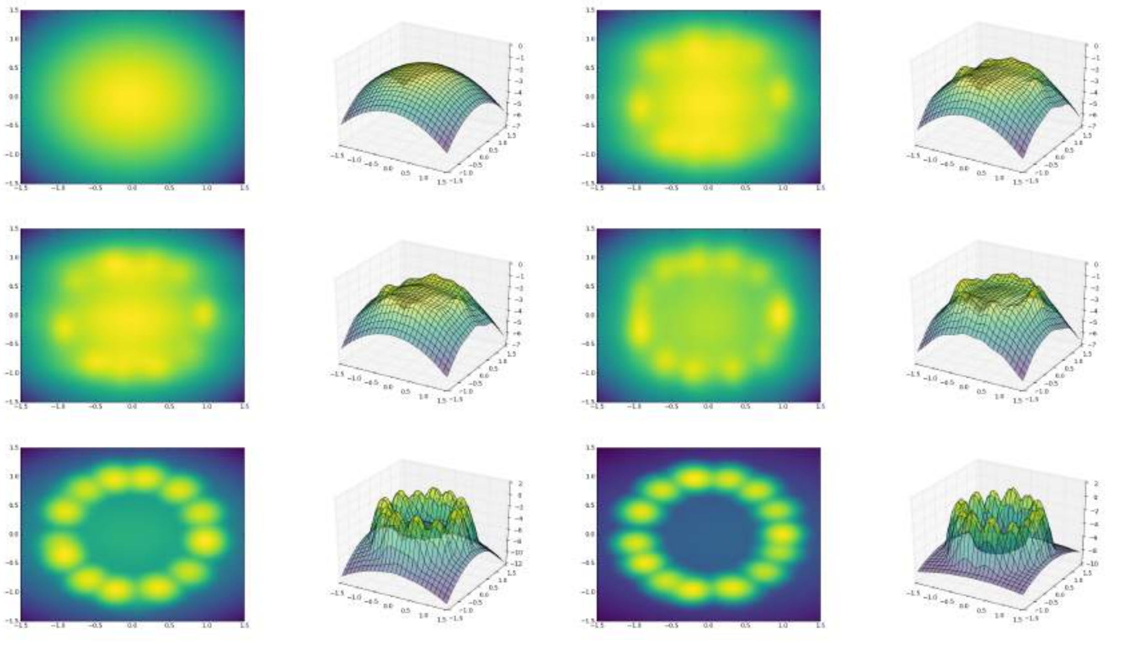

(b) The heatmaps show the evolution of CrossCat's posterior predictive density with increasing number of inference transitions, given a ring with fixed noise level (sixth ring from right in panel (a)). Brighter shades of green indicate greater density mass in the region. The surface plots to the right of each heatmap show the same density, projected in three dimensions. During early stages of inference, the density surface is unimodal and appears as a cloud in the 2D plane. Modalities and patterns in the data are captured with increasing inference, as the Markov chain centers on regions of high posterior mass in CrossCat's latent state.

Figure 3: Using `simulate` and `logpdf` to study CrossCat's emulation of a noisy ring.

4.3 Ensemble classifiers and regressors

In this section, we describe how to construct CGPMs for a class of ensemble- based classifiers and regressors that are common in machine learning. These CGPMs are not typically described by a graphical model (Section 3.3) yet are still able to satisfy the CGPM interface by implementing `simulate` and `logpdf`. For each member r, we assume the CGPM $\mathcal{G}$ generates a single output variable x_r, and requires as input a feature vector $\boldsymbol{y}_r = (y_{[r,1]}, \ldots, y_{[r,I]})$. In an ensemble method, $\mathcal{G}$ carries a set of learners $\{L_1, \ldots, L_K\}$, where each learner L_k returns a point prediction of x_r given $\boldsymbol{y}_r$ denoted $L_k(\boldsymbol{y}_r)$. As a simple example, $\mathcal{G}$ may represent a random forest, and each learner L_i a constituent decision tree. For `infer`, $\mathcal{G}$ may construct the ensemble of learners given measurements $\mathcal{D}$ using any meta-learning algorithm such Boosting (Freund and Schapire, 1995), Bagging (Breiman, 1996) or others.

4.3.1 Classification

Let $\{1, \ldots, S\}$ denote the set of possible values for the output variable x_r (this specification is consistent with a `BINARY` or `NOMINAL` statistical data type from Table 2 in Section 5.3). Given an input $\boldsymbol{y}_r$, the simplest strategy to define a probability for the event $[x_r = s]$ is to compute the proportion of learners in the ensemble who predict $[L_k(\boldsymbol{y}_r) = s]$. This baseline strategy guarantees that the discrete probabilities sum to 1; however, it suffers from degeneracy in that the `simulate` and `logpdf` are undefined when D is empty. To address this issue, we introduce a smoothing parameter α. With probability α, the output x_r is uniform over the S symbols, and with probability $(1-\alpha)$, it is an aggregate of outputs from the learners,

$$p_{\mathcal{G}}(x_r|\boldsymbol{y}_r, \mathcal{D}) = (1-\alpha)\left(\frac{1}{K}\sum_{s=1}^{S}\left(\mathbb{I}[x_r = s]\sum_{k=1}^{K}\left(\mathbb{I}[L_k(\boldsymbol{y}_r) = s]\right)\right)\right) + \alpha\left(\frac{1}{S}\right). \tag{3}$$

In practice, a prior is placed on the smoothing parameter $\alpha \sim \textsc{Uniform}([0,1])$, which is transitioned by gridded Gibbs sampling (Ritter and Tanner, 1992) over the prediction likelihood on the measurement set. When the distribution of x_r given $\boldsymbol{y}_r$ is in the hypothesis space of the learners, we expect that $\lim_{n\to\infty} p_{\mathcal{G}}(\alpha|\mathcal{G}, \mathcal{D}_n) = 0$. Both `simulate` and `logpdf` can be implemented directly from (3).

4.3.2 Regression

In the regression setting, the predictions $\{L_k(\boldsymbol{y}_r)\}$ returned by each learner are real-valued, and so the discrete aggregation strategy from (3) does not lead to a well-defined implementation of `logpdf`. Instead, for an input vector $\boldsymbol{y}_r$ the ensemble-based regression CGPM $\mathcal{G}$ first computes the set of predictions $\{L_1(\boldsymbol{y}_r), \ldots L_K(\boldsymbol{y}_r)\}$, and then `incorporate`s them into a primitive univariate CGPM compatible with the statistical type of the output variable, such as a `NORMAL` for `NUMERICAL`, or `LOGNORMAL` for `MAGNITUDE`. This strategy fits a statistical type appropriate noise model based on the variability of responses from the learners, which relates to how noisy the regression is. implementations of `logpdf` and `simulate` are directly inherited from the constructed primitive CGPM.

4.4 Factor analysis & probabilistic PCA

Our development of factor analysis closely follows (Murphy, 2012, Chatper 12); we extend the exposition to describe implementations of `simulate` and `logpdf` for arbitrary patterns of latent and observable variables. Factor analysis is a continuous latent variable model where the vector of output variables $\boldsymbol{x}_r = (x_{[r,1]}, \dots, x_{[r,D]})$ is a noisy linear combination of a set of L basis vectors $\{\boldsymbol{w}_1, \dots, \boldsymbol{w}_L\}$,

$$\boldsymbol{x}_r = \boldsymbol{\mu} + \boldsymbol{w}_1 z_{[r,1]} + \boldsymbol{w}_2 z_{[r,2]} + \dots + \boldsymbol{w}_L z_{[r,L]} + \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim^{\mathcal{G}} \textsc{Normal}(\mathbf{0}, \operatorname{diag}(\psi_1, \dots, \psi_D)). \tag{4}$$

Each basis vector $\boldsymbol{w}_l$ is a D-dimensional vector and the dimension of the latent space L (a hyperparameter) is less than D. The member latents $\boldsymbol{z}_r \in \mathbb{R}^L$ are known as factor scores, and they represent a low-dimensional projection of $\boldsymbol{x}_r$. The global latents are the bases $\mathbf{W} = [\boldsymbol{w}_1 \dots \boldsymbol{w}_L]$, covariance matrix $\boldsymbol{\Psi}$ of the noise $\boldsymbol{\epsilon}$, and mean vector $\boldsymbol{\mu}$ of $\boldsymbol{x}_r$. To specify a generative model, the member-specific latent variables are given a prior $\boldsymbol{z}_r \sim \textsc{Normal}(\mathbf{0}, \mathbf{I})$. Combining this prior with (4) the joint distribution over the latent and observable variables is

$$\boldsymbol{s}_r = \begin{pmatrix} \boldsymbol{z}_r \\ \boldsymbol{x}_r \end{pmatrix} \sim^{\mathcal{G}} \textsc{Normal}\left(\boldsymbol{m} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\mu} \end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{I}_{L\times L} & \mathbf{W}^\top_{L\times D} \\ \mathbf{W}^\top_{D\times L} & (\mathbf{W}\mathbf{W}^\top + \boldsymbol{\Psi})_{D\times D} \end{pmatrix}\right), \tag{5}$$

where we have defined the joint vector $\boldsymbol{s}_r = (\boldsymbol{z}_r, \boldsymbol{x}_r) \in \mathbb{R}^{D+L}$. The CGPM $\mathcal{G}$ implementing factor analysis exposes the member-specific latent variables as output variables. The multivariate normal (5) provides the ingredients for `simulate` and `logpdf` on any pattern of latent and observable variables with `query` $\boldsymbol{s}_{[r,Q]}$ and `evidence` $\boldsymbol{s}_{[r,E]}$. To arrive at the target distribution, the Bayes theorem for Gaussians (Bishop, 2006) is invoked in a two-step process.

$$\underline{\text{Marginalize}} \qquad \boldsymbol{s}_{[r,Q\cup E]} \sim^{\mathcal{G}} \textsc{Normal}\left(\begin{pmatrix} \boldsymbol{\mu}_Q \\ \boldsymbol{\mu}_E \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_Q & \boldsymbol{\Sigma}_{Q\cup E} \\ \boldsymbol{\Sigma}^\top_{Q\cup E} & \boldsymbol{\Sigma}_E \end{pmatrix}\right)$$

$$\underline{\text{Condition}} \qquad \boldsymbol{s}_{[r,Q]} | \boldsymbol{s}_{[r,E]} \sim^{\mathcal{G}} \textsc{Normal}\left(\boldsymbol{\mu}_Q + \boldsymbol{\Sigma}_{Q\cup E}\boldsymbol{\Sigma}_E^{-1}(\boldsymbol{s}_{[r,E]} - \boldsymbol{\mu}_E), \boldsymbol{\Sigma}_Q - \boldsymbol{\Sigma}_{Q\cup E}\boldsymbol{\Sigma}_E^{-1}\boldsymbol{\Sigma}^\top_{Q\cup E}\right)$$

Our implementation of `infer` uses expectation maximization for factor analysis (Ghahramani and Hinton, 1997); an alternative approach is posterior inference in the Bayesian setting (Press et al., 1997). Finally, probabilistic principal component analysis (Tipping and Bishop, 1999) is recovered when covariance of $\boldsymbol{\epsilon}$ is further constrained to satisfy $\psi_1 = \dots = \psi_D$.

```
%mml CREATE TABLE iris FROM 'iris.csv';
%mml CREATE POPULATION p FOR iris (GUESS (*));
%mml CREATE METAMODEL m FOR p (
....    OVERRIDE GENERATIVE MODEL FOR
....        sepal_length, sepal_width,
....        petal_length, petal_width
....    AND EXPOSE
....        flower_pc1 NUMERICAL,
....        flower_pc2 NUMERICAL
....    USING probabilistic_pca(L=2));
%mml INITIALIZE 1 MODEL FOR m;
%mml ANALYZE m FOR 10 ITERATION;
%bql .scatter
....    INFER EXPLICIT
....        PREDICT flower_pc1 USING 10 SAMPLES,
....        PREDICT flower_pc2 USING 10 SAMPLES,
...         flower_name
....    FROM p;
```

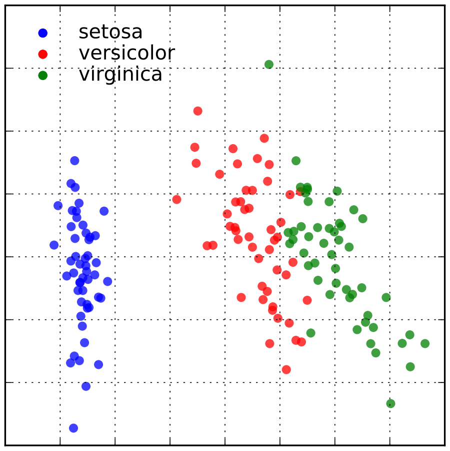


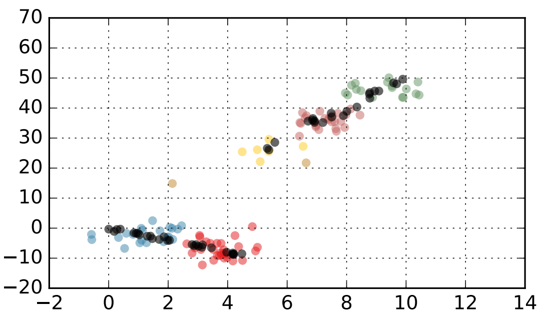
Figure 4: Low dimensional projection of flowers in the iris dataset using the probabilistic PCA CGPM. The two latent principal components scores are exposed as queryable outputs in BQL.

4.5 Parametric mixture of experts

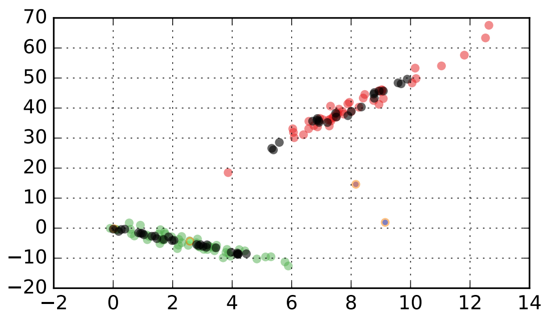
The mixture of experts (Jacobs et al., 1991) is a regression model for data which exhibit highly non-linear characteristics, such as heteroskedastic noise and piecewise continuous patterns. Let $\mathcal{G}$ be a CGPM which generates output variables $\boldsymbol{x}_r = (x_{[r,1]}, \dots, x_{[r,T]})$ given input variables $\boldsymbol{y}_r$, using mixtures of local parametric mixtures. The member latent variable $\boldsymbol{z}_r = (z_r)$ takes values in $[K]$ (possibly unbounded) which induces a Naive Bayes factorization over the outputs

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{y}_r, \boldsymbol{\theta}) = \sum_{k=1}^{K} \left(\prod_{t=1}^{T} p_{\mathcal{G}}(x_{[r,t]}|\boldsymbol{y}_r, \gamma_{[q,z_r]}) p_{\mathcal{G}}(z_r = k|\boldsymbol{y}_r, \boldsymbol{\theta}) \right), \tag{6}$$

where $\gamma_{[q,k]}$ are the regression parameters for variable $x_{[r,t]}$ when $z_r = k$. While (6) looks similar to the Naive Bayes factorization (2) from CrossCat, they differ in important ways. In CrossCat, the variables $x_{[r,t]}$ are sampled from primitive univariate CGPMs, while in the mixture of experts they are sampled from a discriminative CGPM conditioned on $\boldsymbol{y}_r$. The term $p_{\mathcal{G}}(x_{[r,t]}|\boldsymbol{y}_r, \gamma_{[q,z_r]})$ may be any generalized linear model for the correct statistical data type (such as a Gaussian linear regression for `NUMERICAL`, logistic regression for `NOMINAL`, or Poisson regression for `COUNTS`). Second, the mixture of experts has a "gating function" for $p_{\mathcal{G}}(z_r = k|\boldsymbol{y}_r, \boldsymbol{\theta})$ which is also conditioned on $\boldsymbol{y}_r$ and may be a general function such as a softmax or even a Dirichlet process mixture (Hannah et al., 2011). In, CrossCat the member latents $z_{[r,B]}$ are necessarily given a CRP prior in each block. We leave out implementations of `simulate` and `logpdf`, and refer to Figure 5 for a comparison of posterior samples from CrossCat and mixture of experts given data from a piecewise continuous function.



(a) CrossCat



(b) Mixture of linear regression experts

Figure 5: Posterior samples from CrossCat and mixture of experts given a piecewise continuous linear function. Observed data points are shown in black, and posterior samples are shown in color, which represents a latent cluster assignment internal to each CGPM. **(a)** CrossCat emulates the curve using a mixture of axis-aligned Gaussians, requiring a larger number of small, noisy clusters. **(b)** Mixture of linear regression experts identifies the two linear regimes and is able to interpolate well (red dots in top curve). The two orange datapoints that appear as outliers are samples from a "singleton" cluster, since the gating function is implemented using a Dirichlet process mixture.

4.6 Generative nearest neighbors

In this section, we present a compositional generative population model which implements `simulate` and `logpdf` by building ad-hoc statistical models on a per-query basis. The method is a simple extension of K Nearest Neighbors to generative modeling.

Let $\mathcal{G}$ be a generative nearest neighbor CGPM, and $\boldsymbol{x}_{[r,Q]}$ and $\boldsymbol{x}_{[r,E]}$ denote the query and evidence for a `simulate` or `logpdf` query. The method first finds the K nearest neighbors to r in dataset $\mathcal{D}$, based on the values of the evidence variables $\boldsymbol{x}_{[r,E]}$. Let $\mathcal{N}$ denote the top K neighbors, whose generic member is denoted $\boldsymbol{x}_k \in \mathcal{N}$. Within $\mathcal{N}$, we assume the query variables Q are independent, and learn a CGPM $\mathcal{G} = \{\mathcal{G}_{[q]} : q \in Q\}$ which is a product of primitive univariate CGPMs $\mathcal{G}_q$ (based on the appropriate statistical data type of each variable q from Table 2). The measurements $\{x_{[k,q]} k \in \mathcal{N}\}$ are used to learn the primitive CGPM for q in the neighborhood. This procedure is summarized in Algorithm 3c. Implementations of `simulate` and `logpdf` follow directly from the product CGPM, as summarized in Algorithms 3a and 3b. Figure 6 illustrates how the behavior of `simulate` on a synthetic x-cross varies with the neighborhood size parameter K.

It should be noted that building independent models in the neighborhood will result in very poor performance when the query variables remain highly correlated even when conditioned on the evidence. Our baseline approach can be modified to capture the dependence between the query variables by instead building one independent CGPM around the local neighborhood of each neighbor $k \in \mathcal{N}$, rather than one independent CGPM for the entire neighborhood. These improvements are left for future work.

Algorithm 3a `simulate` for generative nearest neighbors CGPM.

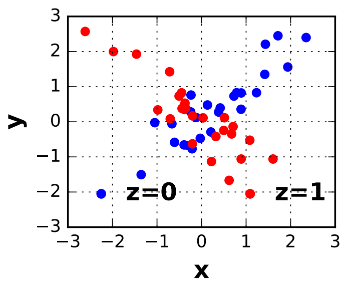
```
1: $\boldsymbol{x}_{[r,Q]} \leftarrow \varnothing$                                          ▷ initialize empty sample
2: $(\mathcal{G}_q : q \in Q) \leftarrow$ Build-Local-Cgpms $(\boldsymbol{x}_{[r,E]})$          ▷ retrieve the local parametric CGPMs
3: for $q \in Q$ do                                                ▷ for each query variable $q$
4:     $x_{[r,q]} \leftarrow$ simulate$(\mathcal{G}_{[j,q]}, r, \{q\}, \varnothing)$              ▷ sample from the primitive CGPM
5: return $\boldsymbol{x}_{[r,Q]}$                                         ▷ overall sample of query variables
```

Algorithm 3b `logpdf` for generative nearest neighbors CGPM.

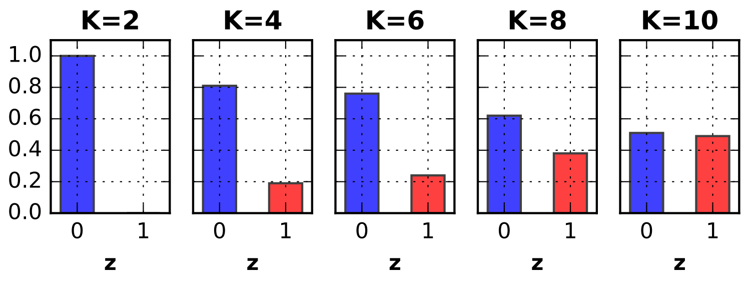
```
1: $(\mathcal{G}_q : q \in Q) \leftarrow$ Build-Local-Cgpms $(\boldsymbol{x}_{[r,E]})$          ▷ retrieve the local parametric CGPMs
2: for $q \in Q$ do                                                ▷ for each query variable $q$
3:     $\log w_q \leftarrow$ logpdf$(\mathcal{G}_q, r, x_{[r,q]}, \varnothing)$                  ▷ compute the density of $q$
4: return $\sum_{q \in Q} \log w_q$                                     ▷ overall density estimate
```

Algorithm 3c Building local parametric models in the generative nearest neighbor CGPM.

```
1: function Build-Local-Cgpms $(\boldsymbol{x}_{[r,E]})$
2:     $\mathcal{D}_E \leftarrow \{\boldsymbol{x}_{[r',E]} : r' \in \mathcal{D}\}$                 ▷ marginalize by exclusion from neighbor search
3:     $\mathcal{N} \leftarrow$ Nearest-Neighbors$(K, \mathcal{D}_E, \boldsymbol{x}_{[r,E]})$                  ▷ find neighbors of $r$
4:     for $q \in Q$ do                                            ▷ for each query variable $q$
5:         $\mathcal{G}_q \leftarrow$ Primitive-Univariate-Cgpm                    ▷ initialize a primitive CGPM
6:         for $k \in \mathcal{N}$ do:                                       ▷ for each neighbor
7:             $\mathcal{G}_q \leftarrow$ incorporate$(\mathcal{G}_q, k, x_{[k,q]})$               ▷ incorporate into primitive CGPM
8:         $\mathcal{G}_q \leftarrow$ infer$(\mathcal{G}_q, \mathcal{T}_{\mathrm{ML}})$                          ▷ transition the primitive CGPM
9:     return $(\mathcal{G}_q : q \in Q)$                                  ▷ collection of primitive CGPMs
```



(a) Observed data



(b) Samples of `z GIVEN x=0.5, y=0.5` for various neighborhood sizes

```
%mml CREATE METAMDOEL xcross_m WITH BASELINE gknn(K=?) FOR xcross;
%bql .scatter SIMULATE z FROM xcross_m GIVEN x=0.5, y=0.5 LIMIT 50;
```

Figure 6: Posterior samples from the generative nearest neighbors CGPM given an x-cross for varying values of neighbors K. (a) Samples from the synthetic x-cross data generator. It produces three variables: `x` and `y` are real-valued and are scattered in the 2D plane, and `z` is a binary variable indicating the functional regime. **(b)** For small neighborhoods (`K`=2, `K`=4), most members of the neighborhood satisfy `z=0`, as reflected by the sharp posterior distribution of `z` at 0. As the neighborhood size increases (`K`=8, `K`=10) they become noisy and include more members with `z=1`, smoothing out the posterior over `z` between 0 and 1.

4.7 Multivariate kernel density estimation

In this section, we show how to express multivariate kernel density estimation with mixed data types, as developed by (Racine and Li, 2004), using CGPMs. Similarly to ensemble methods (Section 4.3) this approach implements the CGPM interface without admitting a natural representation in terms of the graphical model in Figure 1. We extend the exposition of (Racine and Li, 2004) to include algorithms for conditional sampling and density assessment. Given measurements $\mathcal{D}$, the joint distribution over the variables of $\boldsymbol{x}_r$ is estimated non-parametrically

$$p_{\mathcal{G}}(\boldsymbol{x}_r|\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{r'\in\mathcal{D}} \mathcal{K}(\boldsymbol{x}_r|\gamma) = \frac{1}{|\mathcal{D}|}\sum_{r'\in\mathcal{D}}\left(\prod_{i\in[O]}\frac{1}{\gamma_i}K_i\left(x_{[r,i]}, x_{[r',i]}|\gamma_i\right)\right). \quad (7)$$

$\mathcal{K}(\boldsymbol{x}_r|\gamma)$ is a product kernel and γ is a global parameter containing the bandwidths for each kernel K_i. Note that using a product kernel does not imply independence of elements $x_{[r,i]}$ and $x_{[r,j]}$ within a member. Bandwidths are typically learned by cross-validation or maximum-likelihood. For a `NOMINAL` statistical type with S symbols the kernel is

$$K_q(x, x'|\gamma_q) = \Big((1-\gamma_q)\mathbb{I}[x = x'] + \gamma_q/(S-1)\mathbb{I}[x \neq x']\Big),$$

from (Aitchison and Aitken, 1976). For a `NUMERICAL` statistical type the kernel is a standard second order Gaussian

$$K_q(x, x'|\gamma_q) = \left(\exp(-\frac{1}{2}((x-x')/\gamma)^2)/\sqrt{2\pi}\right).$$

To implement `simulate` and `logpdf`, we first show how the product kernel (7) ensures marginalization is tractable,

Marginalize

$$\begin{aligned}
p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\mathcal{D}) &= \int_{\boldsymbol{x}_{[r,\backslash Q]}} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}, \boldsymbol{x}_{[r,\backslash Q]}) d\boldsymbol{x}_{[r,\backslash Q]} = \int_{\boldsymbol{x}_{[r,\backslash Q]}} \frac{1}{|\mathcal{D}|} \sum_{r' \in \mathcal{D}} \mathcal{K}(\boldsymbol{x}_r|\gamma) d\boldsymbol{x}_{[r,\backslash Q]} \\
&= \int_{\boldsymbol{x}_{[r,\backslash Q]}} \left[\frac{1}{|\mathcal{D}|} \sum_{r' \in \mathcal{D}} \left(\prod_{i \in [O]} \frac{1}{\gamma_i} K_i \left(x_{[r,i]}, x_{[r',i]} | \gamma_i \right) \right) d\boldsymbol{x}_{[r,\backslash Q]} \right] \\
&= \frac{1}{|\mathcal{D}|} \sum_{r' \in \mathcal{D}} \left(\int_{\boldsymbol{x}_{[r,\backslash Q]}} \left[\left(\prod_{q \in Q} \frac{1}{\gamma_q} K_q \left(x_{[r,q]}, x_{[r',q]} | \gamma_q \right) \right) \left(\prod_{j \in \backslash Q} \frac{1}{\gamma_j} K_j \left(x_{[r,j]}, x_{[r',j]} | \gamma_j \right) \right) d\boldsymbol{x}_{[r,\backslash Q]} \right] \right) \\
&= \frac{1}{|\mathcal{D}|} \sum_{r' \in \mathcal{D}} \left(\left(\prod_{q \in Q} \frac{1}{\gamma_q} K_q \left(x_{[r,q]}, x_{[r',q]} | \gamma_q \right) \right) \underbrace{\int_{\boldsymbol{x}_{[r,\backslash Q]}} \left[\left(\prod_{j \in \backslash Q} \frac{1}{\gamma_j} K_j \left(x_{[r,j]}, x_{[r',j]} | \gamma_j \right) \right) d\boldsymbol{x}_{[r,\backslash Q]} \right]}_{\text{density normalized to 1}} \right) \\
&= \frac{1}{|\mathcal{D}|} \sum_{r' \in \mathcal{D}} \left(\prod_{q \in Q} \frac{1}{\gamma_q} K_q \left(x_{[r,q]}, x_{[r',q]} | \gamma_q \right) \right). \qquad (8)
\end{aligned}$$

Conditioning is a direct application of the Bayes Rule, where the numerator and denominator are computed separately using (8).

Condition

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, D) = \frac{p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}, \boldsymbol{x}_{[r,E]}|D)}{p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|D)} \qquad (9)$$

Combining (8) and (9) provides an immediate algorithm for `logpdf`. To implement `simulate`, we begin by ignoring the normalizing constant in the denominator of (9) which is unnecessary for sampling. We then express the numerator suggestively,

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \mathcal{D}) \propto \sum_{r' \in \mathcal{D}} \left(\prod_{q \in Q} \frac{1}{\gamma_q} K_q \left(x_{[r,q]}, x_{[r',q]} | \gamma_q \right) \underbrace{\prod_{e \in E} \frac{1}{\gamma_e} K_e \left(x_{[r,e]}, x_{[r',e]} | \gamma_e \right)}_{\text{weight } w'_r} \right), \qquad (10)$$

In particular, the `simulate` algorithm first samples a member $r' \sim \textsc{Categorical}(\{w'_r : r \in \mathcal{D}\})$, where the weight w'_r is labeled in (10). Next, it samples the query elements $x_{[r,q]}$ independently from the corresponding kernels curried at r'. Intuitively, the CGPM weights each member r' in the population by how well its local kernel explains the evidence $\boldsymbol{x}_{[r,E]}$ known about r.

4.8 Probabilistic programs in VentureScript

In this section, we show how to construct a composable generative population model directly in terms of its computational and statistical definitions from Section 3 by expressing it in the VentureScript probabilistic programming language. For simplicity, this section assumes the CGPM satisfies a more refined conditional independence constraint than (1), namely

$$\exists q, q' : (r, c) \neq (r', c') \implies x_{[r,c]} \perp\!\!\!\perp x_{[r',c']} \mid \{\boldsymbol{\alpha}, \boldsymbol{\theta}, z_{[r,q]}, z_{[r',q']}, \boldsymbol{y}_r, \boldsymbol{y}'_r\}. \tag{11}$$

In words, for every observation element $x_{[r,c]}$, there exists a latent variable $z_{[r,q]}$ that (in addition to $\boldsymbol{\theta}$) mediates all coupling with other variables in the population. The member latent variables $\boldsymbol{Z}$ may still exhibit arbitrary dependencies within and among one another. While not essential, this requirement simplifies exposition of the inference algorithms. The approach for `simulate` and `logpdf` is based on approximate inference in tagged subparts of the Venture trace.[3] The CGPM carries a set of K independent samples $\{\boldsymbol{\theta}_k\}_{k=1}^K$ from an approximate posterior $p_{\mathcal{G}}(\boldsymbol{\theta}|\mathcal{D})$. These samples of global latent variables are assigned weights on a per-query basis. Since VentureScript CGPMs are Bayesian, the target distribution for `simulate` and `logpdf` marginalizes over all internal state,

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \mathcal{D}) = \int_{\boldsymbol{\theta}} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}, \mathcal{D}) p_{\mathcal{G}}(\boldsymbol{\theta}|\boldsymbol{x}_{[r,E]}, \mathcal{D}) d\boldsymbol{\theta} \tag{12}$$

$$= \int_{\boldsymbol{\theta}} p(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}, \mathcal{D}) \frac{p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|\boldsymbol{\theta}, \mathcal{D}) p(\boldsymbol{\theta}|\mathcal{D})}{p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|\mathcal{D}, \mathcal{G})} d\boldsymbol{\theta}$$

$$\approx \frac{1}{\sum_{k=1}^K w_k} \sum_{k=1}^K p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}_k, \mathcal{D}) w_k \qquad \boldsymbol{\theta}_k \sim^{\mathcal{G}} |\mathcal{D}. \tag{13}$$

The weight $w_k = p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|\boldsymbol{\theta}_k, \mathcal{D})$ is the likelihood of the evidence under $\boldsymbol{\theta}_k$. The weighting scheme (13) is a computational trade-off circumventing the requirement to run inference on population parameters $\boldsymbol{\theta}$ on a per-query basis, i.e. when given new evidence $\boldsymbol{x}_{[r,E]}$ about r.[4]

It suffices now to consider the target distribution under single sample $\boldsymbol{\theta}_k$:

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}_k, \mathcal{D}) = \int_{\boldsymbol{z}_r} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}, \boldsymbol{z}_r|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}_k, \mathcal{D}) d\boldsymbol{z}_r \tag{14}$$

$$= \int_{\boldsymbol{z}_r} \left[\left(\prod_{q \in Q} p_{\mathcal{G}}(x_{[r,q]}|\boldsymbol{z}_r, \boldsymbol{\theta}_k) \right) p_{\mathcal{G}}(\boldsymbol{z}_r|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}_k, \mathcal{D}) d\boldsymbol{z}_r \right] \tag{15}$$

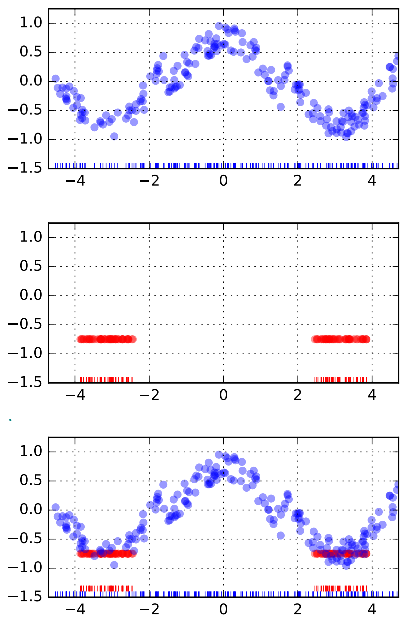
$$\approx \frac{1}{T} \sum_{t=1}^T \prod_{q \in Q} p_{\mathcal{G}}(x_{[r,q]}|\boldsymbol{z}_{[t,r]}, \boldsymbol{\theta}_k) \qquad \boldsymbol{z}_{[t,r]} \sim^{\mathcal{G}} |\{\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}, \mathcal{D}\}. \tag{16}$$

Eq (14) suggests that `simulate` for can be implemented by sampling from the joint local posterior $\{\boldsymbol{x}_{[r,Q]}, \boldsymbol{z}_r|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}_k, \mathcal{D}\}$, and returning only elements $\boldsymbol{x}_{[r,Q]}$. Eq (16) shows that `logpdf` can be implemented by first sampling the member latents $\boldsymbol{z}_r$ from the local posterior. By invoking conditional independence constraint (11) in Eq (15), the query $\boldsymbol{x}_{[r,Q]}$ factors into a product of density terms for each element $x_{[r,q]}$ which can be evaluated directly. This description completes the algorithm for `simulate` and `logpdf` in trace $\boldsymbol{\theta}_k$, and is repeated for $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K\}$. The CGPM implements `simulate` by drawing a trace $j \sim \textsc{Categorical}(\{w_1, \dots, w_K\})$ and returning the sample $\boldsymbol{x}_{[r,Q]}$ from $\boldsymbol{\theta}_j$. Similarly, `logpdf` is computed using the weighted Monte Carlo estimator (13). Algorithms 4a and 4b illustrate implementations in a general probabilistic programming environment.

3. In Venture, every random choice may be in a `scope` which is divided into a set of `blocks`. The CGPM places each member r in its own `scope`, and each observable $x_{[r,i]}$ and latent $z_{[r,i]}$ element in a `block` within that `scope`.
4. An alternative strategy is to compute a harmonic mean estimator based directly on (12).

```
%sql CREATE TABLE sin_t(x, y REAL);
%mml CREATE POPULATION sin_p FOR t WITH SCHEMA(
....   MODEL x, y AS NUMERICAL);

%mml CREATE METAMODEL sin_m FOR sin_p(
....   OVERRIDE MODEL FOR x USING
....     inline_venturescript('
....       () ~> {uniform(low: -4.71, high: 4.71)}
....     ');
....   OVERRIDE MODEL FOR y GIVEN x USING
....     inline_venturescript('
....      (x) ~> {
....        if (cos(x) > 0) {
....          uniform(low: cos(x)-0.5, high: cos(x))
....        else {
....          uniform(low: cos(x), high: cos(x)+0.5)..
....     ')
.... );
%mml ANALYZE 1 MODEL for sin_m;
%bql .scatter SIMULATE x, y FROM sin_p LIMIT 100;
%bql .scatter SELECT x, 0.5 FROM(
....   SIMULATE x FROM sin_p GIVEN y=-0.75 LIMIT 50)
```

(a)



(b)

Figure 7: Composing VentureScript expressions by compiling them into CGPMs. (a) Expressions in teal are lambda expressions, or anonymous functions, in VentureScript, which are compiled into CGPMs by the `inline_venturescript` adapter. Both forward simulation (blue query) and inversion (red query) of the joint generative model are achieved by Algorithm 5a. This code is an instance of polyglot probabilistic programming; it includes expressions from two different languages interacting in a single program. **(b)** The top plot shows samples of forward simulating `x` and `y` (blue query); the middle plot shows samples of `x GIVEN y=-0.75` (red query), successfully capturing the two posterior modes; the bottom plot shows an overlay.

Parameter	Symbol	Parameter	Symbol
no. of trace instances	K	weight of trace k	w_k
global latent variables in trace k	$\boldsymbol{\theta}_k$	sample of $\boldsymbol{z}_r$ in trace k	$\boldsymbol{z}_{[k,r]}$
local latent variables in trace k	$\boldsymbol{Z}_k$	sample of $\boldsymbol{x}_{[r,Q]}$ in trace k	$\boldsymbol{x}_{[k,r,Q]}$
observation set in trace k	$\mathcal{D}_k$	no. of internal Monte Carlo samples	T
input variable	$\boldsymbol{y}_r$	t-th Monte Carlo sample of $\boldsymbol{z}_{[k,r]}$	$\boldsymbol{z}_{[k,t,r]}$
evidence set	$\boldsymbol{x}_{[r,E]}$	weighted density estimate in trace k	q_k

Table 3: Parameters and symbols used in Algorithms 4a and 4b.

Algorithm 4a `simulate` for CGPMs in a general probabilistic programming environment.

```
1: function SIMULATE
2:     for $k = 1, \ldots, K$ do                                  ▷ for each trace $k$
3:         $w_k \leftarrow$ COMPUTE-TRACE-WEIGHT $(k, \boldsymbol{x}_{[r,E]})$     ▷ retrieve the weight
4:     $j \sim$ CATEGORICAL$(\{w_1, \ldots, w_k\})$                ▷ importance resample the traces
5:     $\{\boldsymbol{x}_{[j,r,Q]}, \boldsymbol{z}_{[j,r]}\} \sim^{\mathcal{G}} |\{\boldsymbol{\theta}_j, \boldsymbol{Z}_j, \mathcal{D}_j\}$     ▷ transition operator leaving target invariant
6:     return $\boldsymbol{x}_{[j,r,Q]}$                          ▷ select samples of query set from resampled trace
```

Algorithm 4b `logpdf` for CGPMs in a general probabilistic programming environment.

```
1: function LOGPDF
2:     for $k = 1, \ldots, K$ do                                  ▷ for each trace $k$
3:         $w_k \leftarrow$ COMPUTE-TRACE-WEIGHT $(k, \boldsymbol{x}_{[r,E]})$     ▷ retrieve the weight
4:         for $t = 1, \ldots, T$ do                              ▷ obtain $T$ samples of latents in scope $r$
5:             $\boldsymbol{z}_{[k,t,r]} \sim^{\mathcal{G}} |\{\boldsymbol{\theta}_k, \boldsymbol{Z}_k, \mathcal{D}_k\}$     ▷ transition operator leaving target invariant
6:             $h_{[k,t]} \leftarrow \prod_{q \in Q} p(x_{[r,q]} | \boldsymbol{\theta}_k, \boldsymbol{z}_{[k,t,r]})$     ▷ compute a density estimate
7:         $r_k \leftarrow \frac{1}{T} \sum_{t=1}^{T} h_{[k,t]}$      ▷ aggregate density estimates by simple Monte Carlo
8:         $q_k \leftarrow r_k w_k$                               ▷ importance weight the estimate
9:     return $\log\left(\sum_{k=1}^{K} q_k\right) - \log\left(\sum_{k=1}^{K} w_k\right)$     ▷ weighted importance sampling estimator
```

Algorithm 4c Computing the weight of a trace on a per-query basis.

```
1: function COMPUTE-TRACE-WEIGHT (trace: $k$, evidence: $\boldsymbol{x}_{[r,E]}$)
2:     $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \boldsymbol{y}_r$            ▷ observe the input variable
3:     if $\boldsymbol{z}_{[k,r]} \notin \boldsymbol{Z}_k$ then                 ▷ if member $r$ has unknown local latents
4:         $\boldsymbol{z}_{[k,r]} \sim^{\mathcal{G}} |\{\boldsymbol{\theta}_k, \boldsymbol{Z}_k, \mathcal{D}_k\}$     ▷ sample from the prior
5:     $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \boldsymbol{x}_{[r,E]}$       ▷ observe new evidence variables
6:     $w_k \leftarrow \prod_{e \in E} p_{\mathcal{G}}(x_{[r,e]} | \boldsymbol{\theta}_k, \boldsymbol{z}_{[k,r]})$     ▷ weight by likelihood of $\boldsymbol{x}_{[r,E]}$
7:     return $w_k$
```

5. Integrating Conditional Generative Population Models into BayesDB

Without probabilistic programming systems and languages that treat data analysis computationally, it is difficult to both utilize the expressive power of CGPMs and use general-purpose inference machinery to develop and query them. In this section, we show how CGPMs have been integrated into BayesDB, a probabilistic programming platform with two languages: the Bayesian Query Language (BQL) for model-independent querying, and the Metamodeling Language (MML) for model discovery and building. We first describe how simple BQL queries map directly to invocations of the CGPM interface. We then show how to compose CGPMs into networks, and outline new expressions in MML used to construct populations and networks of CGPMs. The experiments in Section 6 illustrate how extending BayesDB with CGPMs can be used for non-trivial data analysis tasks.

```
%mml CREATE TABLE t FROM "customers.csv"
%mml CREATE POPULATION p FOR t(
....   GUESS STATTYPES FOR (*);
....   MODEL age AS MAGNITUDE
.... );

%mml CREATE METAMODEL m FOR p
....  WITH BASELINE crosscat(
....   SET CATEGORY MODEL
....     FOR age TO lognormal;
....   OVERRIDE GENERATIVE MODEL
....     FOR income GIVEN age, state
....     USING linear_regression
.... );

%mml INITIALIZE 4 MODELS FOR m;
%mml ANALYZE m FOR 1 MINUTE;

%bql SIMULATE age, state
....   GIVEN income = 145000
....   FROM p LIMIT 100;
```

age	state	income
29	CA	145000
61	TX	145000
48	MA	145000

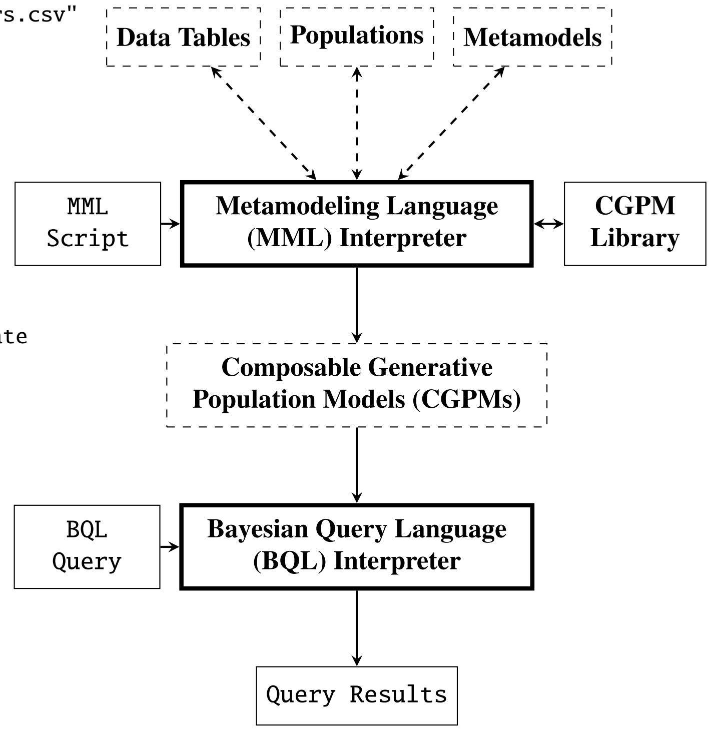

Data Tables
Populations
Metamodels
MML Script
Metamodeling Language (MML) Interpreter
CGPM Library
Composable Generative Population Models (CGPMs)
BQL Query
Bayesian Query Language (BQL) Interpreter
Query Results

Figure 8: System architecture and modules that comprise BayesDB. The Metamodeling Language interpreter reads (i) population schemas to define variables and statistical types, (ii) metamodel definitions to apply automatic and custom modeling strategies for groups of variables in the population, and (iii) commands such as `INITIALIZE`, which instantiates an ensemble of CGPM networks, and `ANALYZE`, which applies inference operators to CGPMs to learn from observed data. The Bayesian Query Language is a model-independent probabilistic query language that allows users to (i) `ESTIMATE` properties of CGPMs such strength and existence of dependence relationships between variables, similarity between members, and conditional density queries, and (ii) `SIMULATE` missing or hypothetical observations subject to user-provided constraints. Together, these components allow users to build population models and query the probable implications of their data.

5.1 Querying composable generative population models using the Bayesian Query Language

The BQL interpreter allows users to ask probabilistic questions about populations using a structured query language. Figure 9 shows how the BQL queries `SIMULATE` and `ESTIMATE PROBABILITY OF` translate into invocations of `simulate` and `logpdf` for an illustrative population and CGPM.

BQL defines a large collection of row-wise and column-wise estimators for CGPMs (Mansinghka et al., 2015a, Sec. 3.2.2), such as `MUTUAL INFORMATION`, `DEPENDENCE PROBABILITY` and `SIMILIARITY WITH RESPECT TO`. These quantities admit default implementations in terms of Monte Carlo estimators formed by `simulate` and `logpdf`, and any CGPM may override the BQL interpreter's generic implementations with a custom, optimized implementation. A full description of implementing BQL in terms of the CGPM interface is beyond the scope of this work.

rowid	a	b	c	d
1	57	2.5	Male	15
2	15	0.8	Female	10
3	NA	1.4	NA	NA
⋮	⋮	⋮	⋮	⋮
r	$x_{[r,a]}$	$x_{[r,b]}$	$x_{[r,c]}$	$y_{[r,d]}$
⋮	⋮	⋮	⋮	⋮

(a) A population represented as an infinite table in BayesDB, modeled by a CGPM $\mathcal{G}$ which generates variables a, b, and c as outputs, and requires variable d as input.

BQL	`SIMULATE a FROM G GIVEN d=12 WHERE rowid=3 LIMIT 2`
CGPM	`simulate` ($\mathcal{G}$, `member`: 3, `query`: {a}, `evidence`: {(d, 12)})
Quantity	$s_i \sim^{\mathcal{G}} x_{[3,a]} \vert \{y_{[3,d]} = 3, x_{[3,b]} = 1.4, \mathcal{D}\}$ for $i = 1, 2$
Result	see table below

rowid	a	d
3	51	12
3	59	12

(b) Mapping a `SIMULATE` query to the CGPM interface invocation of `simulate`. The sampled quantity s_i also includes $\{x_{[3,b]} = 1.4\}$ as a conditioning value, which was extracted from the dataset $\mathcal{D}$. The CGPM must condition on every observed value in $\mathcal{D}$, as well as additional per-query constraints specified by the user, such as $\{y_{[3,d]} = 3\}$. The result is a table with two rows corresponding to the two requested samples.

BQL	`ESTIMATE PROBABILITY OF a=49, c='MALE' GIVEN d=12 FROM G WHERE rowid=3`
CGPM	`logpdf(` $\mathcal{G}$, `member`: 3, `query` : {(a, 49), (c, `'MALE'`)} `evidence` : {(d, 12)})
Quantity	$p_{\mathcal{G}}(x_{[3,a]} = 49, x_{[3,c]} = \text{'MALE'} \vert y_{[3,d]} = 12, x_{[3,b]} = 1.4, \mathcal{D})$
Result	see table below

rowid	a	c	d	bql_pdf((a,c),(d))
3	49	'Male'	12	0.117

(c) Mapping an `ESTIMATE PROBABILITY OF` query to the CGPM interface invocation of `logpdf`. The output is a table with a single row that contains the value of the queried joint density.

Figure 9: Translating BQL queries into invocations of the CGPM interface.

5.2 Compositional networks of composable generative population models

Our development of CGPMs has until now focused on their computational interface and their internal probabilistic structures. In this section, we outline the mathematical formalism which justifies closure of CGPMs under input/output composition. For a collection of CGPMs $\{\mathcal{G}_k : k \in [K]\}$ operating on the same population $\mathcal{P}$, we will show how they be organized into a generalized directed graph which itself is a CGPM $\mathcal{G}_{[K]}$, and provide a Monte Carlo strategy for performing joint inference over the outputs and inputs to the internal CGPMs. This composition allows complex probabilistic models to be built from simpler CGPMs. They communicate with one another using the `simulate` and `logpdf` interface to answer queries against the overall network. In the next section, we describe the surface syntaxes in MML to construct networks of CGPMs in BayesDB.

Let $\boldsymbol{v} = (v_1, \dots, v_T)$ be the variables of $\mathcal{P}$, and $\mathcal{G}_a$ be a CGPM which generates outputs $\boldsymbol{v}_a^{out} = (v_{[a,1]}^{out}, \dots, v_{[a,O_a]}^{out})$, accepts inputs $\boldsymbol{v}_a^{in} = (v_{[a,1]}^{in}, \dots, v_{[a,I_a]}^{in})$, and satisfies $(\boldsymbol{v}_a^{out} \cup \boldsymbol{v}_a^{in}) \subset \boldsymbol{v}$. Similarly, consider another CGPM $\mathcal{G}_b$ on the same population with outputs $\boldsymbol{v}_b^{out}$ and inputs $\boldsymbol{v}_b^{in}$. The composition $(\mathcal{G}_{[b,\mathcal{B}]} \circ \mathcal{G}_{[a,\mathcal{A}]})$ applies the subset of outputs $\boldsymbol{v}_{[a,\mathcal{A}]}^{out}$ of $\mathcal{G}_a$ to the subset of inputs $\boldsymbol{v}_{[b,\mathcal{B}]}^{in}$ of $\mathcal{G}_b$, resulting in a new CGPM $\mathcal{G}^c$ with output $(\boldsymbol{v}_a^{out} \cup \boldsymbol{v}_b^{out})$ and input $(\boldsymbol{v}_a^{in} \cup \boldsymbol{v}_{[b,\backslash\mathcal{B}]}^{out})$. The rules of composition require that $(\boldsymbol{v}_a^{out} \cap \boldsymbol{v}_b^{out}) = \varnothing$ i.e. $\mathcal{G}_a$ and $\mathcal{G}_b$ do not share any output, and that $\boldsymbol{v}_{[a,\mathcal{A}]}^{out}$ and $\boldsymbol{v}_{[b,\mathcal{B}]}^{in}$ correspond to the same subset of variables in the original population $\mathcal{P}$. Generalizing this idea further, a collection of CGPMs $\{\mathcal{G}_k : k \in [K]\}$ can thus be organized as a graph where node k represents internal CGPM $\mathcal{G}_k$, and the labeled edge $a_{\mathcal{A}} \to b_{\mathcal{B}}$ denotes the composition $(\mathcal{G}_{[b,\mathcal{B}]} \circ \mathcal{G}_{[a,\mathcal{A}]})$. These labeled edges between different CGPMs in the network must form a directed acyclic graph. However, elements $x_{[k,r,i]}$ and $x_{[k,r,j]}$ of the same member r within any particular $\mathcal{G}_k$ are only required to satisfy constraint (1) which may in general follow directed and/or undirected dependencies. The topology of the overall CGPM network $\mathcal{G}_{[K]}$ can be summarized by its generalized adjacency matrix $\pi_{[K]} := \{\pi_k : k \in [K]\}$, where $\pi_k = \{(p,t) : v_{[p,t]}^{out} \in \boldsymbol{v}_k^{in}\}$ is the set of all output elements from upstream CGPMs connected to the inputs of $\mathcal{G}_k$.

To illustrate that the class of CGPMs is closed under composition, we need to show how the network $\mathcal{G}_{[K]}$ implements the interface. First note that $\mathcal{G}_{[K]}$ produces as `outputs` the union of all output variables of its constituent CGPMs, and takes as `inputs` the collection of variables in the population which are not the output of any CGPM in the network. The latter collection of variables are "exogenous" to the network, and must be provided for queries that require them.

The implementations of `simulate` and `logpdf` against $\mathcal{G}_{[K]}$ are shown in Algorithms 5a and 5b. Both algorithms use an importance sampling scheme which combines the methods provided by each individual node $\mathcal{G}_k$, and a shared forward-sampling subroutine in Algorithm 5c. The estimator for `logpdf` uses ratio likelihood weighting; both estimators derived from lines 2 and 4 of Algorithm 5b are computed using unnormalized importance sampling, so the ratio estimator on line 6 is exact in the infinite limit of importance samples J and J'. The algorithms explicitly pass the member id r between each CGPM so that they agree about which member-specific latent variables are relevant for the query, while preserving abstraction boundaries. The importance sampling strategy used for compositional `simulate` and `logpdf` may only be feasible when the networks are shallow and the primitive CGPMs are fairly noisy; better Monte Carlo strategies or perhaps even variational strategies may be needed for deeper networks, and are left to future work.

The network's `infer` method can be implemented by invoking `infer` separately on each internal CGPM node. In general, several improvements on this baseline strategy are possible and are also interesting areas for further research (Section 7).

Parameter	Symbol
number of importance samples	J, J'
identifier of the population	r
indices of CGPM nodes in the network	$k = 1, 2, \ldots, K$
CGPM representing node k	$\mathcal{G}_k$
parents of node k	π_k
input variables exogenous to network for node k	$\boldsymbol{y}_{[k,r]}$
query set for node k	$\boldsymbol{x}_{[k,r,Q_k]}$
evidence set for node k	$\boldsymbol{x}_{[k,r,E_k]}$
query/evidence sets aggregated over all nodes in network	$\boldsymbol{x}_{[r,A]} = \bigcup_{k \in [K]} \boldsymbol{x}_{[k,r,A_k]}$

Table 4: Parameters and symbols used in Algorithms 5a, 5b, and 5c. All parameters provided to the functions in which they appear. Weighted-Sample ignores `query` and `evidence` from the global environment, and is provided with an explicit set of constrained nodes by Simulate and LogPdf.

Algorithm 5a `simulate` in a directed acyclic network of CGPMs.

```
1: function Simulate
2:     for $j = 1, \ldots, J$ do                                   ▷ generate $J$ importance samples
3:         $(\boldsymbol{s}_j, w_j) \leftarrow$ Weighted-Sample $(\boldsymbol{x}_{[r,E]})$          ▷ retrieve sample weighted by evidence
4:     $m \leftarrow$ Categorical$(\{w_1, \ldots, w_J\})$                       ▷ resample importance sample
5:     return $\bigcup_{k \in [K]} \boldsymbol{x}_{[k,r,Q_k]} \in \boldsymbol{s}_m$                 ▷ overall sample of query variables
```

Algorithm 5b `logpdf` in a directed acyclic network of CGPMs.

```
1: function LogPdf
2:     for $j = 1, \ldots, J$ do                                   ▷ generate $J$ importance samples
3:         $(\boldsymbol{s}_j, w_j) \leftarrow$ Weighted-Sample $(\boldsymbol{x}_{[r,E]} \cup \boldsymbol{x}_{[r,Q]})$     ▷ joint density of query/evidence
4:     for $j = 1, \ldots, J'$ do                                  ▷ generate $J'$ importance samples
5:         $(\boldsymbol{s}'_j, w'_j) \leftarrow$ Weighted-Sample $(\boldsymbol{x}_{[r,E_k]})$              ▷ marginal density of evidence
6:     return $\log\left(\sum_{[J]} w_j / \sum_{[J']} w_j\right) - \log(J/J')$      ▷ likelihood ratio importance estimator
```

Algorithm 5c Weighted forward sampling in a directed acyclic network of CGPMs.

```
1: function Weighted-Sample (constraints: $\boldsymbol{x}_{[r,C_k]}$)
2:     $(\boldsymbol{s}, \log w) \leftarrow (\varnothing, 0)$                          ▷ initialize empty sample with zero weight
3:     for $k \in$ TopoSort $(\{\pi_1 \ldots \pi_K\})$ do                    ▷ topologically sort the adjacency matrix
4:         $\tilde{\boldsymbol{y}}_{[k,r]} \leftarrow \boldsymbol{y}_{[k,r]} \cup \{x_{[p,r,t]} \in \boldsymbol{s} : (p,t) \in \pi_k\}$           ▷ retrieve required inputs at node $k$
5:         $\log w \leftarrow \log w +$ logpdf$(\mathcal{G}_k, r, \boldsymbol{x}_{[k,r,C_k]}, \tilde{\boldsymbol{y}}_{[k,r]})$     ▷ update weight by constraint likelihood
6:         $\boldsymbol{x}_{[k,r,\backslash C_k]} \leftarrow$ simulate$(\mathcal{G}_k, r, \backslash C_k, \boldsymbol{x}_{[k,r,C_k]} \cup \tilde{\boldsymbol{y}}_{[k,r]})$     ▷ simulate unconstrained nodes
7:         $\boldsymbol{s} \leftarrow \boldsymbol{s} \cup \boldsymbol{x}_{[k,r,C_k \cup \backslash C_k]}$                              ▷ append to sample
8:     return $(\boldsymbol{s}, w)$                                       ▷ overall sample and its weight
```

5.3 Building populations and networks of composable generative population models with the Metamodeling Language

As shown in Figure 8, the MML interpreter in BayesDB interacts with data tables and populations, metamodels, and a library of CGPMs. Population schemas are MML programs which are used to declare a list of variables and their statistical types. Every population is backed by a base table in BayesDB, which stores the measurements. Metamodel definitions are MML programs which are used to declare a composite network of CGPMs for a given population. The internal CGPMs nodes in this network come from the CGPM library available to BayesDB. After declaring a population and a metamodel for it, further MML commands are used to instantiate stochastic ensembles of CGPM networks (`INITIALIZE`), and apply inference operators to them (`ANALYZE`).

In this section, we describe the surface level syntaxes in the Metamodeling Language for population schemas, metamodel definitions, and other MML commands. We also describe how to use the Bayesian Query Language to query ensembles of CGPMs at varying levels of granularity. A formal semantics for MML that precisely describes the relationship between the compositional surface syntax and a network of CGPMs is left for future work.

5.3.1 Population Schemas

A population schema declares a collection of variables and their statistical types.

```
CREATE POPULATION <p> FOR <table> WITH SCHEMA (<schemum>[; ...]);
```

Declares a new population `p` in BayesDB. The token `table` references a database table, which stores the measurements and is known as the base table for `p`.

```
schemum := MODEL <var-names> AS <stat-type>
```

Uses `stat-type` as the statistical data type for all the variables named in `var-names`.

```
schemum := IGNORE <var-names>
```

Excludes `var-names` from the population. This command is typically applied for columns in the base table representing unique names, timestamps, and other metadata.

```
schemum := GUESS STATTYPES FOR (* | <var-names>)
```

Uses existing measurements in the base table to guess the statistical data types of columns in the table. When the argument is (`*`), the target columns are all those which do not appear in `MODEL` or `IGNORE`. When the argument is (`var-names`), only those subset of columns are guessed.

Every column in the base table must have a derivable policy (guess, ignore, or explicitly model with a user-provided statistical data type) from the schema. The statistical data types available in MML are shown in Table 2. The `GUESS` command is implemented using various heuristics on the measurements (such as the number of unique values, sparsity of observations, and SQL `TEXT` columns) and only assigns a variable to either `NOMINAL` or `NUMERICAL`. Using a more refined statistical type for a variable is achieved with an explicit `MODEL...AS` command. Finally, two populations identical same base tables and variables, but different statistical type assignments, are considered distinct populations.

5.3.2 Metamodel Definitions

After creating a population $\mathcal{P}$ in BayesDB, we use metamodel definitions to declare CGPMs for the population. This MML program specifies both the topology and internal CGPM nodes of the network (Section 5.2). Starting with a baseline CGPM at the “root”of the graph, nodes and edges are constructed by a sequence overrides that extract variables from the root node and place them into newly created CGPM nodes. The syntax for a metamodel definition is:

```
CREATE METAMODEL <m> FOR <population> WITH BASELINE <baseline-cgpm>
[(<schemum>[; ...])];
```

Declares a new metamodel `m`. The token `population` references a BayesDB population, which contains a set of variable names and their statistical types and is known as the base population for `m`.

```
baseline-cgpm ::= (crosscat | multivariate_kde | generative_knn)
```

Identifies the automatic model discovery engine, which learns the full joint distribution of all variables in the `population` of `m`. Baselines include Cross-Categorization (Section 4.2), Multivariate Kernel Density Estimation (Section 4.7), or Generative K-Nearest-Neighbors (Section 4.6).

```
schemum := OVERRIDE GENERATIVE MODEL FOR <output-vars>
[GIVEN <input-vars>] [AND EXPOSE (<exposed-var> <stat-type>)[, ...]]
USING <cgpm-name>
```

Overrides `baseline-cgpm` by creating a new node in the CGPM network. The node generates `output-vars`, possibly requires the specified `input-vars`. Additionally, the CGPM may expose some of its latent variable as queryable outputs. The token `cgpm-name` refers to the name of the CGPM which is overriding `baseline-cgpm` on the specified subpart of the joint distribution.

```
schemum := SET CATEGORY MODEL FOR <output-var> TO <primitive-cgpm-name>
```

(This command is only available when `baseline-cgpm` is `crosscat`.)

Replaces the default category model used by `crosscat` for `output-var`, based on its statistical type, with an alternative `primitive-cgpm` that is also applicable to that statistical type (last column of Table 2).

To answer arbitrary BQL queries about a population, BayesDB requires each CGPM to carry a full joint model over all the population variables. Thus, each metamodel is declared with a baseline CGPM, such as CrossCat, a non-parametric Bayesian structure learner for high-dimensional and heterogeneous data tables (Mansinghka et al., 2015b), among others outlined in Section 4. It is important to note that the `input-vars` in the `OVERRIDE MODEL` command may be the outputs of not only the baseline but any collection of upstream CGPMs. It is also possible to completely override the baseline by overriding all the variables in the population.

5.3.3 Homogeneous Ensembles of CGPM Networks

In BayesDB, a metamodel $\mathcal{M}$ is formally defined as an ensemble of CGPM networks $\{(\mathcal{G}_k, w_k)\}_{i=1}^{N}$, where w_k is the weight of network $\mathcal{G}_k$ (Mansinghka et al., 2015a, Section 3.1.2). The CGPMs in $\mathcal{M}$ are homogeneous in that (from the perspective of MML) they have the same metamodel definition, and (from the perspective of the CGPM interface) they are all `created` with the same `population`, `inputs`, `outputs`, and `binary`. The ensemble $\mathcal{M}$ is populated with K instances of CGPMs using the following MML command:

```
INITIALIZE <K> MODELS FOR <metamodel>;
```

Creates K independent replicas of the composable generative population model network contained in the MML definition of `metamodel`.

CGPM instances in the ensemble are different in that BayesDB provides each $\mathcal{G}_k$ a unique `seed` during `create`. This means that invoking `infer`($\mathcal{G}_k$, `program`: $\mathcal{T}$) causes each network's internal state to evolve differently over the course of inference (when $\mathcal{T}$ contains non-deterministic execution). In MML surface syntax, `infer` is invoked using the following command:

```
ANALYZE <metamodel> FOR <K> (ITERATIONS | SECONDS) [(<plan>)];
```

Runs analysis (in parallel) on all the initialized CGPM networks in the ensemble, according to an optional inference `plan`.

```
plan := (VARIABLES | SKIP) <var-names>
```

If `VARIABLES`, then runs analysis on all the CGPM nodes which have at least one output variable in `var-names`. If `SKIP`, then then transitions all the CGPM nodes except those which have a an output variable in `var-names`. As outlined at the end of Section 5.2, each CGPM node is learned independently at present time.

Weighted ensembling of homogeneous CGPMs can be interpreted based on the modeling and inference tactics internal to a CGPM. For example, in Bayesian CGPM network where **`ANALYZE`** invokes MCMC transitions, each $\mathcal{G}_k$ may represent a different posterior sample; for variational inference, each $\mathcal{G}_k$ may converge to a different set of latent parameters due to different random initializations. More extensive syntaxes for inference plans in MML are left for future work.

5.3.4 Heterogeneous Ensembles of CGPM Networks

Section 5.3.3 defined a metamodel $\mathcal{M}$ as an ensemble of homogeneous CGPM networks with the same metamodel definition. It is also possible construct a heterogeneous ensemble of CGPM networks by defining a set of metamodels $\{\mathcal{M}_1, \dots, M_K\}$ for the same population $\mathcal{P}$ but with different metamodel definitions. Let $\mathcal{G}_{[k,t]}$ be the t^{th} CGPM network in the metamodel $\mathcal{M}_k$. The Bayesian Query Language is able to query CGPM networks at three levels of granularity, starting from the most coarse to the most granular.

```
(ESTIMATE | SIMULATE | INFER) <bql-expression> FROM <population>;
```

Executes the BQL query by aggregating responses from all metamodels $\{\mathcal{M}_1, \dots, \mathcal{M}_k\}$ defined for <`population`>.

```
(ESTIMATE | SIMULATE | INFER) <bql-expression> FROM <population>
MODELED BY <metamodel-k>;
```

Executes the BQL query by aggregating responses from all the CGPM networks $\{\mathcal{G}_{[k,t]}\}$ that have been initialized with the MML definition for `<metamodel-k>`.

```
(ESTIMATE | SIMULATE | INFER) <bql-expression> FROM <population>
MODELED BY <metamodel-k> USING MODEL <t>;
```

Executes the BQL query by returning the single response from $\mathcal{G}_{[k,t]}$ in `<metamodel-k>`.

Monte Carlo estimators obtained by `simulate` and `logpdf` remain well-defined even when the ensemble contains heterogeneous CGPMs. All CGPMs across different metamodels are defined for the same population, which determines the statistical types of the variables. This guarantees that the associated supports and (product of) base measures (from Table 2) for `simulate` and `logpdf` queries are all type-matched.

5.4 Composable generative population models generalize and extend generative population models in BayesDB

It is informative to compare both the conceptual and technical differences between generative population models (GPMs) in BayesDB (Mansinghka et al., 2015a) with composable generative population models (CGPMs). In its original presentation, the GPM interface served the purpose of being the primary vehicle for motivating BQL as a model-independent query language (Mansinghka et al., 2015a, Sec.3.2). Moreover, GPMs were based around CrossCat as the baseline model-discovery engine (Mansinghka et al., 2015a, Sec. 4.5.1), which provided good solutions for several data analysis tasks. However, by not accepting inputs, GPMs offered no means of composition; non-CrossCat objects, known as “foreign predictors”, were discriminative models embedded directly into the CrossCat joint density (Mansinghka et al., 2015a, Sec. 4.4.2). By contrast, the main purpose of the CGPM interface is to motivate more expressive MML syntaxes for building hybrid models, comprised of arbitrary generative and discriminative components. Since CGPMs natively accept inputs, they admit a natural form of composition (Section 5.2) which does violate the internal representation of any particular CGPM.

The computational interface and probabilistic structure of GPMs and CGPMs are different in several respects. Because GPMs were presented as Bayesian models with Markov Chain Monte Carlo inference (Mansinghka et al., 2015a, Sec. 4.2), both `simulate` and `logpdf` were explicitly conditioned on a particular set of latent variables extracted from some state in the posterior inference chain (Mansinghka et al., 2015a, Sec. 3.1.1). On the other hand, CGPMs capture a much broader set of model classes, and `simulate` and `logpdf` do not impose any conditioning constraints internal to the model besides conditioning on input variables and the entire dataset $\mathcal{D}$. Internally, GPMs enforced much stronger assumptions regulating inter-row independences; all the elements in a row are conditionally independent give a latent variable (Mansinghka et al., 2015a, Sec.3.1), effectively restricting the internal structure to a directed graphical model. CGPMs allow for arbitrary coupling between elements within a row from Eq (1), which uniformly expresses both directed and undirected probabilistic models, as well approaches which are not naturally probabilistic that implement the interface. Finally, unlike GPMs, CGPMs may expose some of member-specific latent variables as queryable outputs. This features trades-off the model independence of BQL with the ability to learn and query the details of the internal probabilistic process encapsulated by the CGPM.

6. Applications of Composable Generative Population Models

The first part of this section outlines a case study applying compositional generative population models in BayesDB to a population of satellites maintained by the Union of Concerned Scientists. The dataset contains 1163 entries, and each satellites has 23 numerical and categorical features such as its material, functional, physical, orbital and economic characteristics. We construct a hybrid CGPM using an MML metamodel definition which combines (i) a classical physics model written as a probabilistic program in VentureScript, (ii) a random forest to classify a a nominal variable, (iii) an ordinary least squares regressor to predict a numerical variable, and (iv) principal component analysis on the real-valued features of the satellites. These CGPMs allow us to identify satellites that probably violate their orbital mechanics, accurately infer missing values of anticipated lifetime, and visualize the dataset by projecting the satellite features into two dimensions.

The second part of this section explores the efficacy of hybrid compositional generative population models on a collection of common tasks in probabilistic data analysis by reporting lines of code and accuracy measurements against standard baseline solutions. Large savings in lines of code and improved accuracy are demonstrated in several important regimes. Most of the analysis of experimental results is contained in the figure gallery at the end of the section.

6.1 Analyzing satellites using a composite CGPM built from causal probabilistic programs, discriminative machine learning, and Bayesian non-parametrics

The left panel in Figure 10 illustrates a session in MML which declares the population schema for the satellites data, as well as the metamodel definition for building the hybrid CGPM network that models various relationships of interest between variables.[5] The `CREATE POPULATION` block shows the high-dimensional features of each satellite and their heterogeneous statistical types. For simplicity, several variables such as `perigee_km`, `launch_mass_kg` and `anticipated_lifetime` have been modeled as `NUMERICAL` rather than a more refined type such as `MAGNITUDE`. In the remainder of this section, we explain the CGPMs declared in the MML metamodel definition under the `CREATE METAMODEL` block, and refer to figures for results of BQL queries executed against them.

The PCA CGPM on line 34 of the metamodel definition generates as output five real-valued variables, and exposes the first two principal component scores to BayesDB. This low-dimensional projection allows us to both visualize a clustering of the dataset in latent space, and discover oddities in the distribution of latent scores for satellites whose `class_of_orbit` is `elliptical`. It also identifies a single satellite, in cyan at grid point $(1, 1.2)$, as a candidate for further investigation. Figure 12 shows the result and further commentary on this experiment.

Four variables in the population relate to the orbital characteristics of each satellite: `apogee_km` A, `perigee_km` P, `period_minutes` T, and `eccentricity` e. These variables are constrained by the theoretical Keplerian relationships $e = \frac{A-P}{A+P}$ and $T = 2\pi\sqrt{\frac{((A+P)/2)^3}{GM}}$, where GM is a physical constant. In reality, satellites deviate from their theoretical orbits for a variety of reasons, such orbital and measurement noise, having engines, or even data-entry errors. The right panel of Figure 10 shows a CGPM in pure VentureScript which accepts as input $\boldsymbol{y}_r = (A_r, P_r)$ (apogee and perigee), and generates as output $x_r = T_r$ (period). The prior is a Dirichlet process mixture model on the

5. This program is executed in iVenture, an experimental interactive probabilistic programming environment that supports running `%bql`, `%mml` and `%venturescript` code cells, all of which operate on a common underlying BayesDB instance and Venture interpreter.

error, based on a stochastic variant of Kepler's Law,

$$\begin{aligned}
&G \sim DP(\alpha, \text{Normal-Inverse-Gamma}(m, V, a, b)) \\
&(\mu_r, \sigma_r^2)|G \sim G \\
&\epsilon_r|\boldsymbol{y}_r \sim \text{Normal}(\cdot|\mu_r, \sigma_r^2) \qquad \text{where } \epsilon_r := T_r - \text{Kepler}(A_r, P_r).
\end{aligned}$$

While the internal details, external interface, and adapter which compiles the VentureScript source into a CGPM are beyond the scope of this paper, note that its MML declaration uses the `EXPOSE` command on line 45. This command makes the inferred cluster identity and noise latent variables (lines 17 and 22 of the VentureScript program) available to BQL. Figure 11 shows a posterior sample of the cluster assignments and error distribution, which identifies three distinct classes of anomalous satellites based on the magnitude of error. For instance, satellite `Orion6` in the right panel of Figure 11, belongs to a cluster with "extreme" deviation. Further investigation reveals that `Orion6` has a period 23.94 minutes, a data-entry error for the true period of 24 hours (1440 minutes).

Figure 13 shows the improvement in prediction accuracy achieved by the hybrid CGPM over the purely generative CrossCat baseline, for a challenging multiclass classification task. As shown in lines 57-62 of the metamodel definition in Figure 10, the hybrid CGPM uses a random forest CGPM for the target variable `type_of_orbit` given five numerical and categorical predictors. Figures 13a and 13b shows the confusion matrices on the test set for both the composite and baseline CGPMs. While both methods systematically confuse sun-synchronous with intermediate orbits, the use of a random forest classifier results in 11 less classification errors, or an improvement of 11 percentage points. Using a purely discriminative model for this task, i.e. a random forest without a generative model over the features (not shown), would require additional logic and heuristic imputation on feature vectors in the test set, which general contained missing entries.

The final experiment in Figure 14 compares the posterior distribution of the vanilla CrossCat baseline and multivariate KDE for a two-dimensional density estimation task with nominal data types. The task is to jointly simulate the `country_of_operator` and `purpose` for a hypothetical satellite, given that its `type_of_orbit` is geosynchronous. The empirical conditional distribution from the dataset is shown in red. Both CrossCat and multivariate KDE capture the posterior modes, although the distribution form KDE has a fatter tail, as indicated by the high number of samples classified as "Other". The figure caption contains additional discussion.

There dozens of additional BQL queries that can be posed about the satellites population and, based on the analysis task of interest, answered using both the existing CGPMs in the hybrid metamodel as well as more customized CGPMs. The empirical studies in this section has shown it is possible and practical to apply CGPMs in BayesDB to challenging data analysis tasks in a real-world dataset, and use BQL queries to compare their performance characteristics.

6.2 Comparing code length and accuracy on representative data analysis tasks

One of the most sparsely observed variables in the satellites dataset is the `anticipated_lifetime`, with roughly one in four missing entries. The analysis task in Figure 15 is to infer the anticipated lifetime x_* of a new satellite, given the subset of its numerical and nominal features $\boldsymbol{y}_*$ shown in the codeblock above the plot. To quantify performance, the predictions of the CGPM were evaluated on a held-out set of satellites with known lifetimes. Many satellites in both the training set and test set contained missing entries in their covariates, requiring the CGPM to additionally impute missing values in the predictors before forward simulating the regression. Unlike the purely generative and purely discriminative baselines (shown in the legend), the hybrid CGPM learns both a joint distribution over the predictors and a discriminative model for the response, leading to significantly improved predictive performance.

The improvement in lines of code over the baseline methods in Figure 15 is due to using combinations of (i) SQL for data processing, (ii) MML for model building, and (iii) BQL for predictive querying, in BayesDB. All the baselines required custom logic for (i) manual data preprocessing such as reading csv files, (ii) Euclidean embedding of large categorical values, and (iii) heuristic imputation of missing features during train and test time (i.e. either imputing the response from its mean value, or imputing missing predictors from their mean values). The left panel from Figure 15a shows and end-to-end session in BayesDB which preprocesses the data, builds the hybrid CGPM, runs analysis on the training set and computes predictions on the test set. The right panel from Figure 15b shows a single ad-hoc routine used by the Python baselines, which dummy codes a data frame with missing entries and nominal data types. For nominal variables taking values in a large set, dummy coding with zeros may cause the solvers to fail when the system is under-determined. The workaround in the code for baselines is to drop such problematic dimensions from the feature vector. The regression in the hybrid CGPM does not suffer from this problem because, the default linear regressor in the CGPM library gives all parameters a Bayesian prior (Banerjee, 2008), which smooths irregularities.

Figures 16, 17, 18 and 19 extend the lines of code and accuracy comparisons for CGPMs and baseline methods to several more tasks using diverse statistical methodologies. These figures further illustrate coverage and conciseness of CGPMs – the captions detail the setup and commentary of each experiment in greater detail.

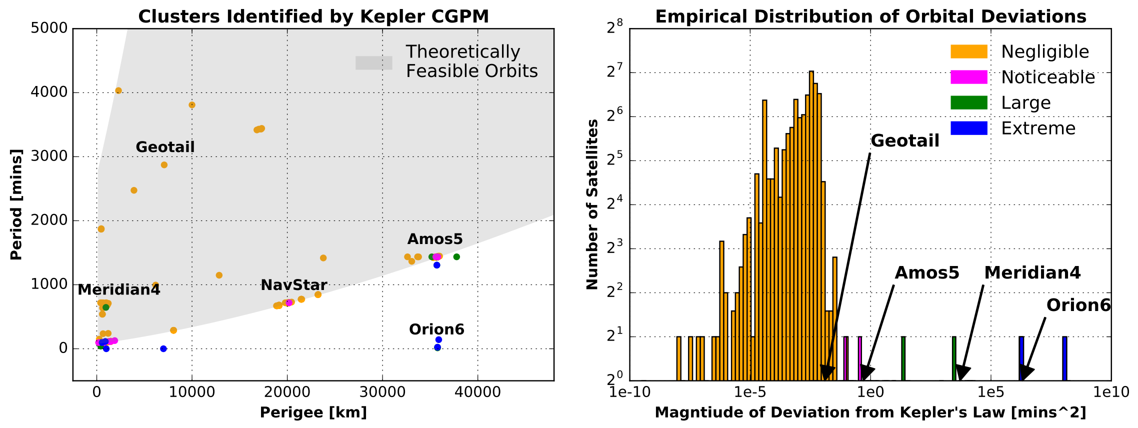
Figure 10: Building a hybrid CGPM in Venturescript and MML for the satellites population.

%mml

```
CREATE TABLE satellites_ucs FROM 'satellites.csv'

.nullify satellites_ucs 'NaN'

CREATE POPULATION satellites FOR satellites_ucs
  WITH SCHEMA (
    IGNORE Name;

    MODEL
      country_of_operator, operator_owner,
      purpose, class_of_orbit, type_of_orbit
      users, contractor, launch_vehicle,
      country_of_contractor, launch_site,
      source_used_for_orbital_data
    AS NOMINAL;

    MODEL
      perigee_km, apogee_km, eccentricity,
      period_minutes launch_mass_kg,
      dry_mass_kg, power_watts,
      date_of_launch, anticipated_lifetime
    AS NUMERICAL;

    MODEL
        longitude_radians_of_geo,
        inclination_radians
    AS CYCLIC
);

CREATE METAMODEL sat_hybrid FOR satellites
  WITH BASELINE crosscat(
    SET CATEGORY MODEL FOR eccentricity TO beta;

    OVERRIDE GENERATIVE MODEL FOR
      launch_mass_kg, dry_mass_kg, power_watts,
      perigee_km, apogee_km
    AND EXPOSE
      pc1 NUMERICAL, pc2 NUMERICAL
    USING factor_analysis(L=2);

    OVERRIDE GENERATIVE MODEL FOR
      period_minutes
    GIVEN
      apogee_km, perigee_km
    AND EXPOSE
      kepler_cluster CATEGORICAL,
      kepler_noise NUMERICAL
    USING venturescript(sp=kepler);

    OVERRIDE GENERATIVE MODEL FOR
      anticipated_lifetime
    GIVEN
      date_of_launch, power_watts, apogee_km,
      perigee_km, dry_mass_kg, class_of_orbit
    USING linear_regression;

    OVERRIDE GENERATIVE MODEL FOR
      type_of_orbit
    GIVEN
      apogee_km, perigee_km, period_minutes,
      users, class_of_orbit
    USING random_forest(k=7);
);
```

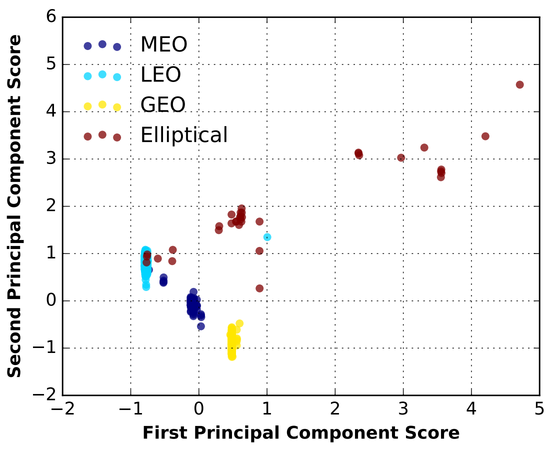
%venturescript

```
// Kepler CGPM.
define kepler = () -> {
  // Kepler's law.
  assume keplers_law = (apogee, perigee) -> {
    let GM = 398600.4418;
    let earth_radius = 6378;
    let a = (abs(apogee) + abs(perigee)) *
        0.5 + earth_radius;
    2 * 3.1415 * sqrt(a**3 / GM) / 60
  };
  // Internal samplers.
  assume crp_alpha = .5;
  assume cluster_sampler = make_crp(crp_alpha);
  assume error_sampler = mem((cluster) ->
        make_nig_normal(1, 1, 1, 1));
  // Output simulators.
  assume sim_cluster_id =
    mem((rowid, apogee, perigee) ~> {
      tag(atom(rowid), atom(1), cluster_sampler())
  });
  assume sim_error =
    mem((rowid, apogee, perigee) ~> {
      let cluster_id = sim_cluster_id(
        rowid, apogee, perigee);
      tag(atom(rowid), atom(2),
        error_sampler(cluster_id)())
  });
  assume sim_period =
    mem((rowid, apogee, perigee) ~> {
      keplers_law(apogee, perigee) +
        sim_error(rowid, apogee, perigee)
  });
  // List of simulators.
  assume simulators = [
    sim_period, sim_cluster_id, sim_error];
};

// Output observers.
define obs_cluster_id =
  (rowid, apogee, perigee, value, label) -> {
    $label: observe sim_cluster_id(
      $rowid, $apogee, $perigee) = atom(value);
};
define obs_error =
  (rowid, apogee, perigee, value, label) -> {
    $label: observe sim_error(
      $rowid, $apogee, $perigee) = value;
};
define obs_period =
  (rowid, apogee, perigee, value, label) -> {
    let theoretical_period = run(
      sample keplers_law($apogee, $perigee));
    obs_error(
      rowid, apogee, perigee,
      value - theoretical_period, label);
};
// List of observers.
define observers = [
  obs_period, obs_cluster_id, obs_error];
// List of inputs.
define inputs = ["apogee", "perigee"];
// Transition operator.
define transition = (N) -> {mh(default, one, N)};
```



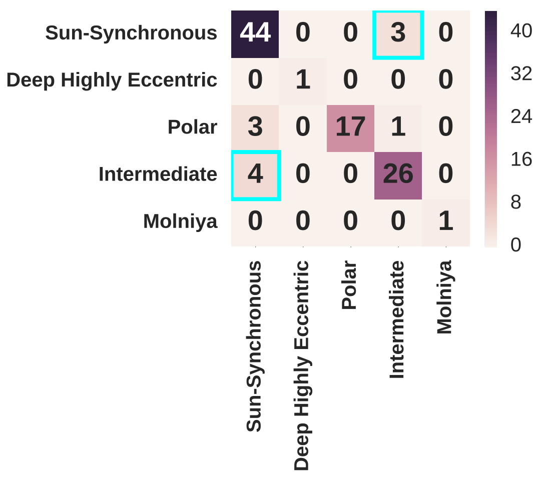
```
%bql INFER kepler_cluster, kepler_noise FROM satellites;
```

Figure 11: Finding satellites whose orbits are likely violations of Kepler's Third Law using a causal CGPM in Venturescript, which learns a Dirichlet process mixture on the residuals. Each dot in the scatter plot (left) is a satellite in the dataset, and its color represents the latent cluster assignment learned by the causal CGPM. Both the cluster identity and inferred noise are exposed latent variables. The histogram (right) shows that each of the four distinct clusters roughly translates to a qualitative description for the magnitude of a satellite's deviation from its theoretical period: yellow (negligible), magenta (noticeable), green (large), and blue (extreme). These clusters were learned non-parametrically.

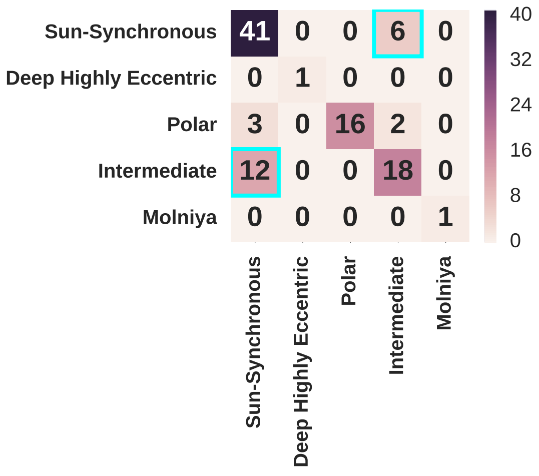


```
%bql INFER EXPLICIT PREDICT pc1, PREDICT pc2, class_of_orbit FROM satellites;
```

Figure 12: Low dimensional projection of the satellites using the PCA CGPM reveals clusterings in latent space and suggests candidate outliers. The principal component scores are based on the numerical features of a satellite, and the color is the `class_of_orbit`. Satellites in low earth, medium earth, and geosynchronous orbit form tight clusters in latent space along PC1, and exhibit most within-cluster variance along PC2. The distribution on factor scores for elliptical satellites has much higher variability along both dimensions, indicating a collection of weak local modes depending on the regime of the satellite's `eccentricity` (not shown), and/or many statistical outliers.



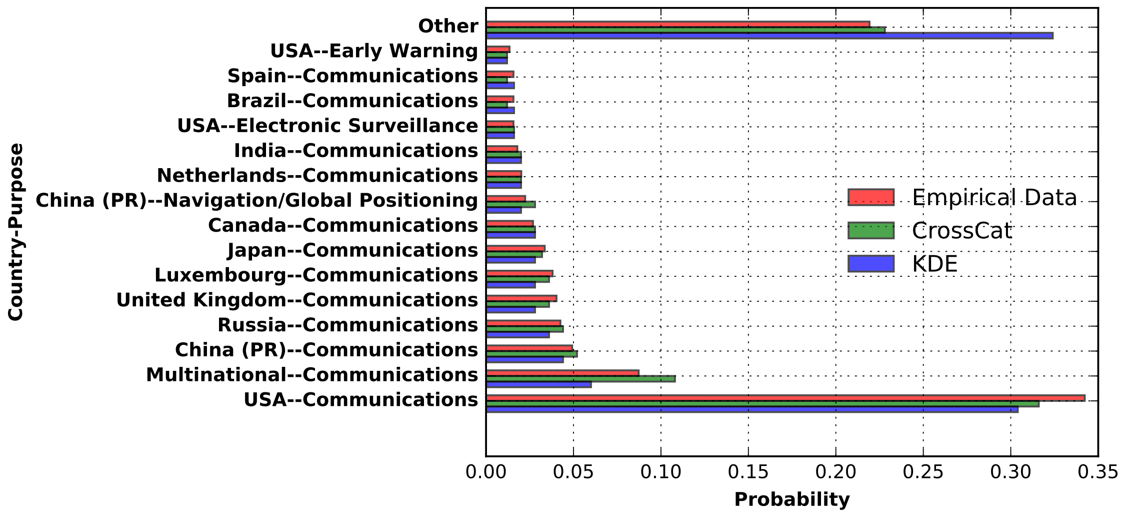
(a) Crosscat/Random Forest hybrid CGPM.



(b) CrossCat baseline CGPM.

```
%bql INFER type_of_orbit FROM held_out_satellites;
```

Figure 13: Confusion matrices for a multiclass classification task show improved prediction accuracy by the hybrid CGPM over the CrossCat baseline. The y-axis shows the true label for "type of orbit" of 100 held-out satellites, and the x-axis shows the predicted label by each CGPM. The feature vectors are five dimensional and consist of numerical and categorical variables (lines 57-62 of Figure 10), and both test and training sets contained missing data. While both CrossCat and Crosscat + Random Forest systematically confuse "sun-synchronous"and "intermediate" orbits (entries in cyan), the overall error rate is reduced by 11% in the hybrid CGPM.



```
%bql SIMULATE country_of_operator, purpose GIVEN class_of_orbit = 'GEO';
```

Figure 14: Simulating from the joint distribution of the country and purpose of a hypothetical satellite, given its orbit type. The y-axis shows the simulated country-purpose pairs, and the x-axis shows the frequency of simulations, compared to the true frequency in the dataset. 500 samples were obtained from CrossCat and multivariate KDE to estimate the posterior probabilities. The posteriors of both CrossCat and KDE are smooth versions of the empirical data – the smoothing for CrossCat is induced by the inner Dirichlet process mixture over category models, and for KDE is induced by the bandwidth parameters of the Aitchison and Aitken kernels. The plot shows that CrossCat's samples provide a tighter fit to the dataset. The distribution from KDE has a fatter tail, as indicated by the high number of samples classified in the "Other" category.

```
CREATE TABLE data_train FROM satellites_train.csv;
.nullify data_train 'NaN';

CREATE POPULATION satellites FOR data_train
  WITH SCHEMA(
    GUESS STATTYPES FOR (*)
);

CREATE METAMODEL cc_ols FOR satellites
  WITH BASELINE crosscat(
    OVERRIDE GENERATIVE MODEL FOR
        anticipated_lifetime
    GIVEN
        type_of_orbit, perigee_km, apogee_km,
        period_minutes, date_of_launch,
        launch_mass_kg
    USING linear_regression
);

INITIALIZE 4 MODELS FOR cc_ols;
ANALYZE cc_ols FOR 100 ITERATION WAIT;

CREATE TABLE data_test FROM satellites_test.csv;
.nullify data_test 'NaN';
.sql INSERT INTO data_train
    SELECT * FROM data_test;

CREATE TABLE predicted_lifetime AS
    INFER EXPLICIT
        PREDICT anticipated_lifetime
        CONFIDENCE pred_conf
    FROM satellites WHERE _rowid_ > 1000;
```

(a) Full session in BayesDB which loads the training and test sets, creates a hybrid CGPM, and runs the regression.

```
def dummy_code_categoricals(frame, maximum=10):

    def dummy_code_categoricals(series):
        categories = pd.get_dummies(
            series, dummy_na=1)
        if len(categories.columns) > maximum - 1:
            return None
        if sum(categories[np.nan]) == 0:
            del categories[np.nan]
        categories.drop(
            categories.columns[-1], axis=1,
            inplace=1)
        return categories

def append_frames(base, right):
    for col in right.columns:
        base[col] = pd.DataFrame(right[col])

numerical = frame.select_dtypes(include=[float])
categorical = frame.select_dtypes(
    include=['object'])
categorical_coded = filter(
    lambda s: s is not None,
    [dummy_code_categoricals(categorical[c])
        for c in categorical.columns])

joined = numerical

for sub_frame in categorical_coded:
    append_frames(joined, sub_frame)

return joined
```

(b) Ad-hoc Python routine (used by baselines) for dummy coding nominal predictors in a dataframe with missing values and heterogeneous types.

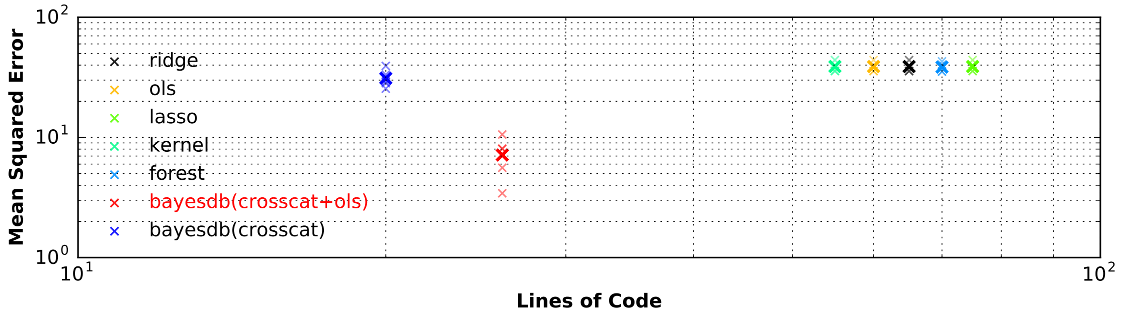
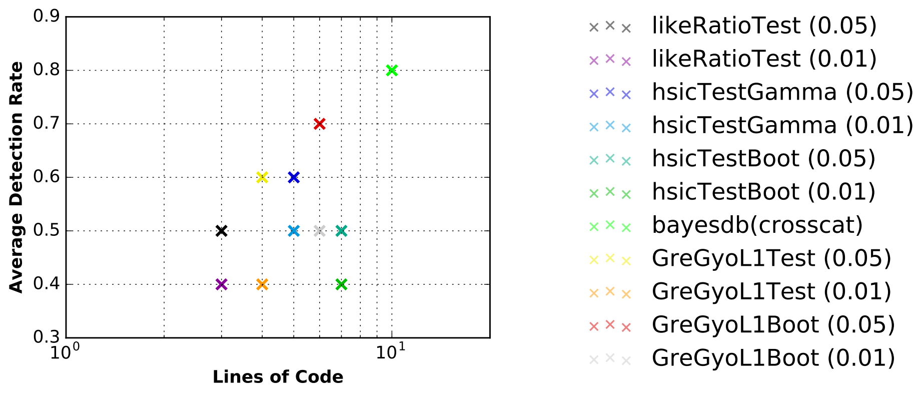
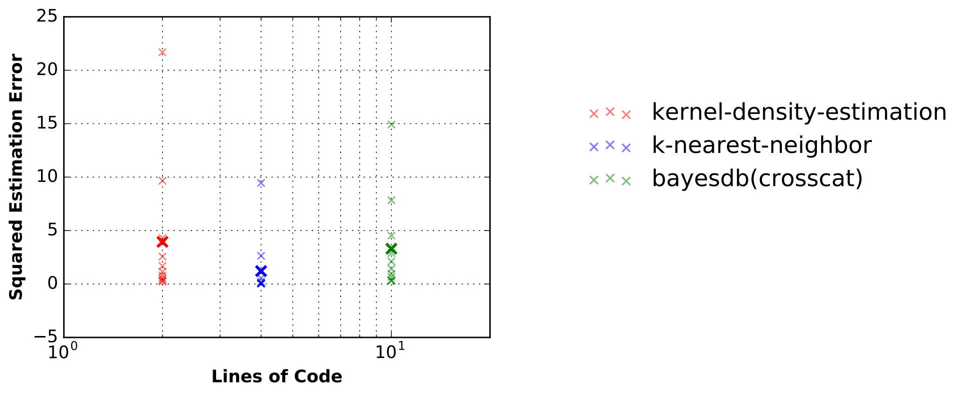


Figure 15: In a high-dimensional regression problem with mixed data types and missing data, the composite CGPM shows improvement in prediction accuracy over purely generative and purely discriminative baselines. The task is to infer the anticipated lifetime of a held-out satellite given categorical and numerical features such as type of orbit, launch mass, and orbital period. Some feature vectors in the test set have missing entries, leading purely discriminative models (ridge, lasso, OLS) to either heuristically impute missing features, or to ignore the features and predict the mean lifetime from its marginal distribution in the training set. The purely generative model (CrossCat) is able to impute missing data from their full joint distribution, but only indirectly mediates dependencies between the predictors and response through latent variables. The composite CGPM (CrossCat+OLS) combines advantages of both approaches; statistically rigorous imputation followed by direct regression on the features leads to improved predictive accuracy.



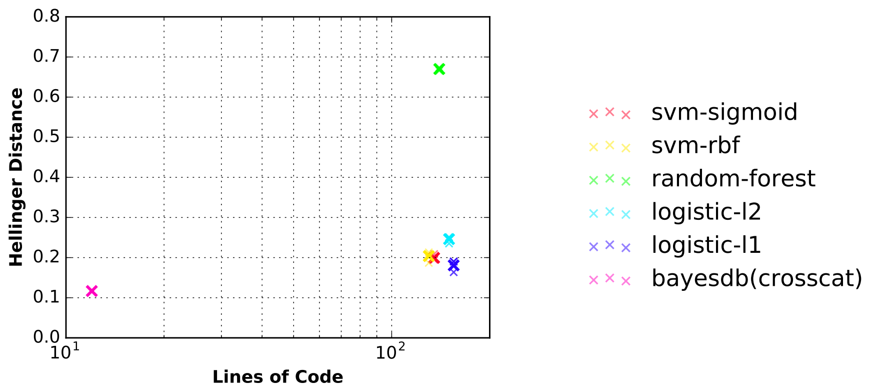
```
%bql ESTIMATE DEPENDENCE PROBABILITY OF x WITH y;
```

Figure 16: Dependence discovery. Binary hypothesis tests of independence for synthetic two-dimensional data drawn from five noisy zero-correlation datasets: sin wave, parabola, x-cross, diamond, and ring. For all datasets the two dimensions are dependent. The y-axis shows the fraction of correct hypotheses achieved by each method, averaged over all datasets. The decision rule for kernel-based tests (Gretton et al., 2007; Gretton and Györfi, 2008, 2010), is based on a frequentist significance level of 5% and 1%. The decision rule for CrossCat is based on a dependence probability threshold of 50%.



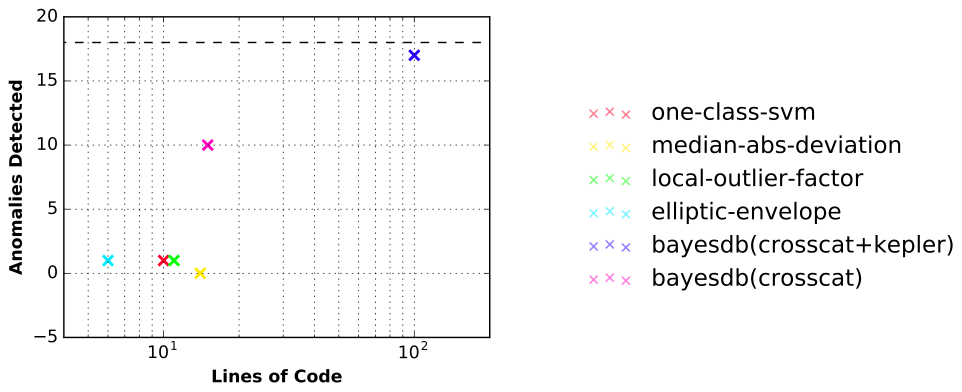
```
%bql ESTIMATE MUTUAL INFORMATION OF x WITH y;
```

Figure 17: Dependence strength Estimating the mutual information of a noisy sin wave. The y-axis shows the squared estimation error, randomized over observed datasets. The "ground truth" mutual information was derived analytically, and the integral computed by quadrature. Baseline methods estimate mutual information using K nearest neighbors (Kraskov et al., 2004) and kernel density estimation (Moon et al., 1995). CrossCat estimates the mutual information first by learning a Dirichlet process mixture of Gaussians, and using Monte Carlo estimation by generating samples from the posterior predictive distribution and assessing their density.



```
%bql SIMULATE country_of_operator, purpose GIVEN class_of_orbit = 'GEO';
```

Figure 18: Bivariate categorical density estimation. Simulating from the posterior joint distribution of the country and purpose of a hypothetical satellite, given its orbit type. 500 samples were obtained from each method to estimate the posterior probabilities. The y-axis shows the Hellinger distance between posterior samples from each method and the empirical conditional distribution from the dataset, used as "ground truth". Standard discriminative baselines struggle to learn the distribution of a two-dimensional discrete outcome based on a discrete input, where both the predictor and response variables take values in large categorical sets.



```
%bql ESTIMATE PREDICTIVE PROBABILITY OF period_minutes;
```

Figure 19: Anomaly detection. Detecting satellites with anomalous orbital periods. 18 satellites from the dataset demonstrated a non-trivial deviation (greater than five minutes) from their theoretical period, used as "ground truth" anomalies. For each method, the top 20 satellites ranked by "outlyingness" score were used as the predicted anomalies. Hybrid CGPMs learn multivariate and multimodal distributions over all variables in the dataset, leading to higher detection rates than baseline methods which use univariate and/or unimodal statistics. The Kepler CGPM identifies most anomalies at the expense of a highly complex program in comparison to baselines.

7. Discussion and Future Work

This paper has shown that it is possible to use a computational formalism in probabilistic programming to apply, combine, and compare a broad class of probabilistic data analysis techniques. CGPMs extend the core provided by directed graphical models, which express elaborate probabilistic models in terms of smaller univariate pieces, by specifying a computational interface that allows these pieces to be multivariate, more black-box, and defined directly as software. A key feature of this framework is that it enables statistical modelers to compose discriminative, generative and hybrid models from different philosophies in machine learning and statistics using probabilistic programming. Moreover, the compositional abstraction is neutral to a CGPM's internal choices of (i) modeling assumptions, which may be i.e. hierarchical or flat, or Bayesian or non-Bayesian, and (ii) inference tactics, which may be i.e. optimization- or sampling-based.

Several models from statistics admit natural implementations in terms of the current CGPM interface, such as non-linear mixed effect models (Davidian and Giltinan, 1995), where each member represents a potentially repeated measurement with latent variables grouping the members into observation units; or Gaussian processes (Rasmussen and Williams, 2006), where the input variables are time indexes from another CGPM, and the outputs are noisy observations of the (latent) function values (Tresp, 2001; Rasmussen and Ghahramani, 2002). Computational representations of these models as CGPMs allows them to be composable as hybrid models, reusable as software, and queryable in interesting ways using the Bayesian Query Language.

Both `simulate` and `logpdf` in Listing 1 are executed against a single member of the population i.e. variables within a single row. Queries that target multiple members in the population are currently supported by an explicit sequence of `incorporate`, `infer`, and then `simulate` or `logpdf`. It is interesting to consider extending the CGPM interface to natively handle arbitrary multi-row cases – this idea was originally presented in the GPM interface (Mansinghka et al., 2015a, Section 3.1.1) although concrete algorithms for implementing multi-row queries, or surface-level syntax in the Bayesian Query Language for invoking them, were left as open questions. Rather than support multi-row queries directly in the CGPM interface, it is instead possible to extend the BQL interpreter with a probabilistic query planner. Given given a cross-row query, the BQL interpreter automatically determines a candidate set of invocation sequences of the CGPM interface to answer it, and then selects among them based on time/accuracy requirements.

A worthy direction for future work is extending the set of statistical data types (Section 4.1), and possibly CGPM interface, to support analysis tasks beyond traditional multivariate statistics. Some possible new data types and associated CGPMs are

- `GRAPH` data type, using a relational data CGPM based on the stochastic block model (Nowicki and Snijders, 2001) or infinite relational model (Kemp et al., 2006),
- `TEXT` data type, using a topic model CGPM such as latent Dirichlet allocation (Blei et al., 2003) or probabilistic latent semantic analysis (Hofmann, 1999),
- `IMAGE` data type, using a CGPM based on neural networks.

Composing CGPMs with these data types leads to interesting tasks over their induced joint distributions. Consider an `IMAGE` variable with an associated `TEXT` annotation; a generative CGPM for the image and discriminative CGPM for the text (given the image) leads to image classification; a generative CGPM for the text and a discriminative CGPM for the image (given the text) allows simulating unstructured text followed by their associated images.

It is also interesting to consider introducing additional structure to our current formalism of populations from Section 3.1 to support richer notions of population modeling. For instance, populations may be hierarchical in that the variables of population A correspond to outputs produced by a CGPM for population B – the simplest case being summary statistics such as means, medians, and inter-quartile ranges. Such hierarchical populations are common in census data, which contain raw measurements of variables for individual households, as well as row-wise and column-wise summaries based on geography, income level, ethnicity, educational background, and so on. Populations can also be extended to support "merge" operations in MML, which are analogous to the `JOIN` operations in SQL, where the CGPM on the joined population allows for transfer learning.

Our presentation of the algorithm for `infer` in a composite network of CGPMs (Section 5.2) left open improvements to the baseline strategy of learning each CGPM node separately. One way to achieve joint learning, without violating the abstraction boundaries of the CGPM interface, is: after running `infer` individually for each CGPM, run a "refine" phase, where (i) missing measurements in the population are imputed using one forward pass of `simulate` throughout the network, then (ii) each CGPM updates its parameters based on the imputed measurements. This strategy can be repeated to generate several such imputed networks, which are then organized into an ensemble of CGPMs in a BayesDB metamodel (Section 5.3.3) where each CGPM in the metamodel corresponds to a different set of imputations. The weighted-averaging of these CGPMs by BayesDB would thus correspond to integration over different imputations, as well as their induced parameters.

Extending BQL, or developing new probabilistic programming languages, to assess the inference quality of CGPMs built in MML will be an important step toward broader application of these probabilistic programming tools for real-world analysis tasks. For instance, it is possible to develop a command in BQL such as

```
ESTIMATE KL DIVERGENCE BETWEEN <cgpm-1> AND <cgpm-2>
  FOR VARIABLES <var-names-a> GIVEN <var-names-b>;
```

which takes two CGPMs (and an overlapping subset of their output variables) and returns an estimate of the KL divergence between their conditional predictive distributions, based on a Monte Carlo estimator using `simulate` and `logpdf`. Such model-independent estimators of inference quality, backed by the CGPM interface, provide a proposal for unifying the testing and profiling infrastructure among a range of candidate solutions for a given data analysis task.

This paper has shown that it is possible to unify and formalize a broad class of probabilistic data analysis techniques by integrating them into a probabilistic programming platform, which is itself integrated with a traditional database. We have focused on a class of probabilistic models that can be tightly integrated with flat database tables. Population schemas define the variables of interest along with their types, but unlike traditional database schemas, they can additionally include variables whose values are never directly observed. Concrete probabilistic models for populations are built via automated inference mechanisms, according to a baseline meta-modeling strategy which can also be customized. This idea is similar to concrete indexes for tables in traditional databases which are built by automated mechanisms, according to an indexing strategy which can be customized via its own schema. While we are encouraged by the early successes of this approach, there is a vast literature of richer "data modeling" formalisms from both databases and statistics. Integrating these ideas could yield further conceptual insight and practical benefits. We hope this paper encourages others to develop these connections, along with a new generation of intelligent tools for machine-assisted probabilistic data analysis.

References

John Aitchison and Colin GG Aitken. Multivariate binary discrimination by the kernel method. *Biometrika*, 63(3):413–420, 1976.

David J Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIIIâĂŤ1983*, pages 1–198. Springer, 1985.

Charles E Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, pages 1152–1174, 1974.

Sudipto Banerjee. *Bayesian Linear Model: Gory Details*, 2008. URL `http://www.biostat.umn.edu/~ph7440/pubh7440/BayesianLinearModelGoryDetails.pdf`.

C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN 9780387310732.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Bob Carpenter, Daniel Lee, Marcus A Brubaker, Allen Riddell, Andrew Gelman, Ben Goodrich, Jiqiang Guo, Matt Hoffman, Michael Betancourt, and Peter Li. Stan: A probabilistic programming language. 2015.

G. Casella and R.L. Berger. *Statistical Inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002. ISBN 9780534243128.

Andreas C Damianou and Neil D Lawrence. Deep gaussian processes. In *AISTATS*, pages 207–215, 2013.

M. Davidian and D.M. Giltinan. *Nonlinear Models for Repeated Measurement Data*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1995. ISBN 9780412983412.

Luc Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pages 260–265. ACM, 1986.

Daniel Fink. A compendium of conjugate priors. 1997.

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

A. Gelman and J. Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Analytical Methods for Social Research. Cambridge University Press, 2006. ISBN 9781139460934.

Zoubin Ghahramani and Geoffrey E. Hinton. The em algorithm for mixtures of factor analyzers. Technical report, 1997.

Andrew D Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. Tabular: a schema-driven probabilistic programming language. In *ACM SIGPLAN Notices*, volume 49, pages 321–334. ACM, 2014.

Arthur Gretton and László Györfi. Nonparametric independence tests: Space partitioning and kernel approaches. In *Algorithmic Learning Theory*, pages 183–198. Springer, 2008.

Arthur Gretton and László Györfi. Consistent nonparametric tests of independence. *The Journal of Machine Learning Research*, 11:1391–1423, 2010.

Arthur Gretton, Kenji Fukumizu, Choon H Teo, Le Song, Bernhard Schölkopf, and Alex J Smola. A kernel statistical test of independence. In *Advances in neural information processing systems*, pages 585–592, 2007.

Lauren A Hannah, David M Blei, and Warren B Powell. Dirichlet process mixtures of generalized linear models. *Journal of Machine Learning Research*, 12(Jun):1923–1953, 2011.

Matthew D Hoffman, David M Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. 2006.

R. Khattree and D.N. Naik. *Multivariate Data Reduction and Discrimination with SAS Software*. Wiley, 2000. ISBN 9780471323006.

Daphne Koller, Nir Friedman, Lise Getoor, and Ben Taskar. 2 graphical models in a nutshell. 2007.

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

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*, 2015a.

Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua B Tenenbaum. Crosscat: A fully bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *arXiv preprint arXiv:1512.01272*, 2015b.

Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. Estimation of mutual information using kernel density estimators. *Physical Review E*, 52(3):2318, 1995.

K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning. MIT Press, 2012. ISBN 9780262304320.

Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

T.D. Nielsen and F.V. Jensen. *Bayesian Networks and Decision Graphs*. Information Science and Statistics. Springer New York, 2009. ISBN 9780387682822. URL `https://books.google.com/books?id=37CAgCykQaAC`.

Krzysztof Nowicki and Tom A B Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455):1077–1087, 2001.

Jim Pitman. Combinatorial stochastic processes. 2002.

S. James Press, S. James Press, K. Shigemasu, and K. Shigemasu. Bayesian inference in factor analysis - revised. Technical report, 1997.

Jeff Racine and Qi Li. Nonparametric estimation of regression functions with both categorical and continuous data. *Journal of Econometrics*, 119(1):99–130, 2004.

Carl Edward Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems*, 2:881–888, 2002.

C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. Adaptative computation and machine learning series. University Press Group Limited, 2006. ISBN 9780262182539.

Christian Ritter and Martin A Tanner. Facilitating the gibbs sampler: the gibbs stopper and the griddy-gibbs sampler. *Journal of the American Statistical Association*, 87(419):861–868, 1992.

Havard Rue and Leonhard Held. *Gaussian Markov random fields: theory and applications*. CRC Press, 2005.

Jayaram Sethuraman. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650, 1994.

N.H. Timm. *Applied Multivariate Analysis*. Springer Texts in Statistics. Springer New York, 2002. ISBN 9780387953472.

Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Volker Tresp. Mixtures of gaussian processes. In *Advances in Neural Information Processing Systems 13*, pages 654–660. MIT Press, 2001.

A Probabilistic Programming Approach To Probabilistic Data Analysis

Feras Saad
MIT Probabilistic Computing Project
fsaad@mit.edu

Vikash Mansinghka
MIT Probabilistic Computing Project
vkm@mit.edu

Abstract

Probabilistic techniques are central to data analysis, but different approaches can be challenging to apply, combine, and compare. This paper introduces composable generative population models (CGPMs), a computational abstraction that extends directed graphical models and can be used to describe and compose a broad class of probabilistic data analysis techniques. Examples include discriminative machine learning, hierarchical Bayesian models, multivariate kernel methods, clustering algorithms, and arbitrary probabilistic programs. We demonstrate the integration of CGPMs into BayesDB, a probabilistic programming platform that can express data analysis tasks using a modeling definition language and structured query language. The practical value is illustrated in two ways. First, the paper describes an analysis on a database of Earth satellites, which identifies records that probably violate Kepler's Third Law by composing causal probabilistic programs with non-parametric Bayes in 50 lines of probabilistic code. Second, it reports the lines of code and accuracy of CGPMs compared with baseline solutions from standard machine learning libraries.

1 Introduction

Probabilistic techniques are central to data analysis, but can be difficult to apply, combine, and compare. Such difficulties arise because families of approaches such as parametric statistical modeling, machine learning and probabilistic programming are each associated with different formalisms and assumptions. The contributions of this paper are (i) a way to address these challenges by defining CGPMs, a new family of composable probabilistic models; (ii) an integration of this family into BayesDB [10], a probabilistic programming platform for data analysis; and (iii) empirical illustrations of the efficacy of the framework for analyzing a real-world database of Earth satellites.

We introduce composable generative population models (CGPMs), a computational formalism that generalizes directed graphical models. CGPMs specify a table of observable random variables with a finite number of columns and countably infinitely many rows. They support complex intra-row dependencies among the observables, as well as inter-row dependencies among a field of latent random variables. CGPMs are described by a computational interface for generating samples and evaluating densities for random variables derived from the base table by conditioning and marginalization. This paper shows how to package discriminative statistical learning techniques, dimensionality reduction methods, arbitrary probabilistic programs, and their combinations, as CGPMs. We also describe algorithms and illustrate new syntaxes in the probabilistic Metamodeling Language for building composite CGPMs that can interoperate with BayesDB.

The practical value is illustrated in two ways. First, we describe a 50-line analysis that identifies satellite data records that probably violate their theoretical orbital characteristics. The BayesDB script builds models that combine non-parametric Bayesian structure learning with a causal probabilistic program that implements a stochastic variant of Kepler's Third Law. Second, we illustrate coverage and conciseness of the CGPM abstraction by quantifying the improvement in accuracy and reduction in lines of code achieved on a representative data analysis task.

2 Composable Generative Population Models

A composable generative population model represents a data generating process for an exchangeable sequence of random vectors $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots)$, called a population. Each member $\boldsymbol{x}_r$ is T-dimensional, and element $x_{[r,t]}$ takes values in an observation space $\mathcal{X}_t$, for $t \in [T]$ and $r \in \mathbb{N}$. A CGPM $\mathcal{G}$ is formally represented by a collection of variables that characterize the data generating process:

$$\mathcal{G} = (\boldsymbol{\alpha}, \boldsymbol{\theta}, \boldsymbol{Z} = \{\boldsymbol{z}_r : r \in \mathbb{N}\}, \boldsymbol{X} = \{\boldsymbol{x}_r : r \in \mathbb{N}\}, \boldsymbol{Y} = \{\boldsymbol{y}_r : r \in \mathbb{N}\}).$$

- $\boldsymbol{\alpha}$: Known, fixed quantities about the population, such as metadata and hyperparameters.
- $\boldsymbol{\theta}$: Population-level latent variables relevant to all members of the population.
- $\boldsymbol{z}_r = (z_{[r,1]}, \dots z_{[r,L]})$: Member-specific latent variables that govern only member r directly.
- $\boldsymbol{x}_r = (x_{[r,1]}, \dots x_{[r,T]})$: Observable output variables for member r. A subset of these variables may be observed and recorded in a dataset $\mathcal{D}$.
- $\boldsymbol{y}_r = (y_{[r,1]}, \dots y_{[r,I]})$: Input variables, such as "feature vectors" in a purely discriminative model.

A CGPM is required to satisfy the following conditional independence constraint:

$$\forall r \neq r' \in \mathbb{N}, \forall t, t' \in [T] : x_{[r,t]} \perp\!\!\!\perp x_{[r',t']} \mid \{\boldsymbol{\alpha}, \boldsymbol{\theta}, \boldsymbol{z}_r, \boldsymbol{z}_{r'}\}. \tag{1}$$

Eq (1) formalizes the notion that all dependencies *across* members $r \in \mathbb{N}$ are completely mediated by the population parameters $\boldsymbol{\theta}$ and member-specific variables $\boldsymbol{z}_r$. However, elements $x_{[r,i]}$ and $x_{[r,j]}$ within a member are generally free to assume any dependence structure. Similarly, the member-specific latents in $\boldsymbol{Z}$ may be either uncoupled or highly-coupled given population parameters $\boldsymbol{\theta}$. CGPMs differ from the standard mathematical definition of a joint density in that they are defined in terms of a computational interface (Listing 1). As computational objects, they explicitly distinguish between the *sampler* for the random variables from their joint distribution, and the *assessor* of their joint density. In particular, a CGPM is required to sample/assess the joint distribution of a subset of output variables $\boldsymbol{x}_{[r,Q]}$ conditioned on another subset $\boldsymbol{x}_{[r,E]}$, and marginalizing over $\boldsymbol{x}_{[r,[T]\setminus(Q \cup E)]}$.

Listing 1 Computational interface for composable generative population models.

- s $\leftarrow$ `simulate` ($\mathcal{G}$, `member`: r, `query`: $Q = \{q_k\}$, `evidence` : $\boldsymbol{x}_{[r,E]}$, `input` : $\boldsymbol{y}_r$)
  Generate a sample from the distribution s $\sim^{\mathcal{G}} \boldsymbol{x}_{[r,Q]} | \{\boldsymbol{x}_{[r,E]}, \boldsymbol{y}_r, \mathcal{D}\}$.
- $c \leftarrow$ `logpdf` ($\mathcal{G}$, `member`: r, `query` : $\boldsymbol{x}_{[r,Q]}$, `evidence` : $\boldsymbol{x}_{[r,E]}$, `input` : $\boldsymbol{y}_r$)
  Evaluate the log density $\log p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]} | \{\boldsymbol{x}_{[r,E]}, \boldsymbol{y}_r, \mathcal{D}\})$.
- $\mathcal{G}' \leftarrow$ `incorporate` ($\mathcal{G}$, `measurement` : $x_{[r,t]}$ or $\boldsymbol{y}_r$)
  Record a measurement $x_{[r,t]} \in \mathcal{X}_t$ (or $\boldsymbol{y}_r$) into the dataset $\mathcal{D}$.
- $\mathcal{G}' \leftarrow$ `unincorporate` ($\mathcal{G}$, `member` : r)
  Eliminate all measurements of input and output variables for member r.
- $\mathcal{G}' \leftarrow$ `infer` ($\mathcal{G}$, `program` : $\mathcal{T}$)
  Adjust internal latent state in accordance with the learning procedure specified by program $\mathcal{T}$.

2.1 Primitive univariate CGPMs and their statistical data types

The statistical data type (Figure 1) of a population variable x_t generated by a CGPM provides a more refined taxonomy than its "observation space" $\mathcal{X}_t$. The (parameterized) support of a statistical type is the set in which samples from `simulate` take values. Each statistical type is also associated with a base measure which ensures `logpdf` is well-defined. In high-dimensional populations with heterogeneous types, `logpdf` is taken against the product measure of these base measures. The statistical type also identifies invariants that the variable maintains. For instance, the values of a `NOMINAL` variable are permutation-invariant. Figure 1 shows statistical data types provided by the Metamodeling Language from BayesDB. The final column shows some examples of primitive CGPMs that are compatible with each statistical type; they implement `logpdf` directly using univariate probability density functions, and algorithms for `simulate` are well known [4]. For `infer` their parameters may be fixed, or learned from data using, e.g., maximum likelihood [2, Chapter 7] or Bayesian priors [5]. We refer to an extended version of this paper [14, Section 3] for using these primitives to implement CGPMs for a broad collection of model classes, including non-parametric Bayes, nearest neighbors, PCA, discriminative machine learning, and multivariate kernel methods.

Statistical Data Type	Parameters	Support	Measure/σ-Algebra	Primitive CGPM
`BINARY`	-	$\{0,1\}$	$(\#, 2^{\{0,1\}})$	`BERNOULLI`
`NOMINAL`	symbols: S	$\{0 \dots S{-}1\}$	$(\#, 2^{[S]})$	`CATEGORICAL`
`COUNT/RATE`	base: b	$\{0, \frac{1}{b}, \frac{2}{b}, \dots\}$	$(\#, 2^{\mathbb{N}})$	`POISSON`, `GEOMETRIC`
`CYCLIC`	period: p	$(0, p)$	$(\lambda, \mathcal{B}(\mathbb{R}))$	`VON-MISES`
`MAGNITUDE`	–	$(0, \infty)$	$(\lambda, \mathcal{B}(\mathbb{R}))$	`LOGNORMAL`, `EXPON`
`NUMERICAL`	–	$(-\infty, \infty)$	$(\lambda, \mathcal{B}(\mathbb{R}))$	`NORMAL`
`NUMERICAL-RANGED`	low: l, high:h	$(l, h) \subset \mathbb{R}$	$(\lambda, \mathcal{B}(\mathbb{R}))$	`BETA`, `NORMAL-TRUNC`

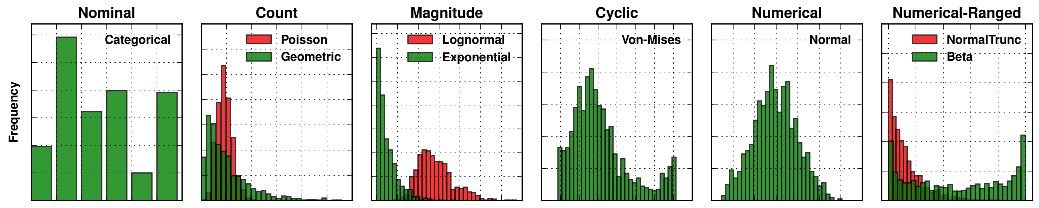


Figure 1: Statistical data types for population variables generated by CGPMs available in the BayesDB Metamodeling Language, and samples from their marginal distributions.

2.2 Implementing general CGPMs as probabilistic programs in VentureScript

In this section, we show how to implement `simulate` and `logpdf` (Listing 1) for composable generative models written in VentureScript [8], a probabilistic programming language with programmable inference. For simplicity, this section assumes a stronger conditional independence constraint,

$$\exists l, l' \in [L] \text{ such that } (r,t) \neq (r',t') \implies x_{[r,t]} \perp\!\!\!\perp x_{[r',t']} \mid \{\boldsymbol{\alpha}, \boldsymbol{\theta}, z_{[r,l]}, z_{[r',l']}, \boldsymbol{y}_r, \boldsymbol{y}'_r\}. \tag{2}$$

In words, for every observable element $x_{[r,t]}$, there exists a latent variable $z_{[r,l]}$ which (in addition to $\boldsymbol{\theta}$) mediates all coupling with other variables in the population. The member latents $\boldsymbol{Z}$ may still exhibit arbitrary dependencies. The approach for `simulate` and `logpdf` described below is based on approximate inference in tagged subparts of the Venture trace, which carries a full realization of all random choices (population and member-specific latent variables) made by the program. The runtime system carries a set of K traces $\{(\boldsymbol{\theta}^k, \boldsymbol{Z}^k)\}_{k=1}^{K}$ sampled from an approximate posterior $p_{\mathcal{G}}(\boldsymbol{\theta}, \boldsymbol{Z}|\mathcal{D})$. These traces are assigned weights depending on the user-specified evidence $\boldsymbol{x}_{[r,E]}$ in the `simulate/logpdf` function call. $\mathcal{G}$ represents the CGPM as a probabilistic program, and the input $\boldsymbol{y}_r$ and latent variables $\boldsymbol{Z}^k$ are treated as ambient quantities in $\boldsymbol{\theta}^k$. The distribution of interest is

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \mathcal{D}) = \int_{\boldsymbol{\theta}} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}, \mathcal{D}) p_{\mathcal{G}}(\boldsymbol{\theta}|\boldsymbol{x}_{[r,E]}, \mathcal{D}) d\boldsymbol{\theta}$$

$$= \int_{\boldsymbol{\theta}} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}, \mathcal{D}) \left(\frac{p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|\boldsymbol{\theta}, \mathcal{D}) p_{\mathcal{G}}(\boldsymbol{\theta}|\mathcal{D})}{p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|\mathcal{D})} \right) d\boldsymbol{\theta} \tag{3}$$

$$\approx \frac{1}{\sum_{k=1}^{K} w^k} \sum_{k=1}^{K} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}) w^k \qquad \text{where } \boldsymbol{\theta}^k \sim^{\mathcal{G}} |\mathcal{D}. \tag{4}$$

The weight $w^k = p_{\mathcal{G}}(\boldsymbol{x}_{[r,E]}|\boldsymbol{\theta}^k, \mathcal{D})$ of trace $\boldsymbol{\theta}^k$ is the likelihood of the evidence. The weighting scheme (4) is a computational trade-off avoiding the requirement to run posterior inference on population parameters $\boldsymbol{\theta}$ for a query about member r. It suffices to derive the distribution for only $\boldsymbol{\theta}^k$,

$$p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}) = \int_{\boldsymbol{z}_r^k} p_{\mathcal{G}}(\boldsymbol{x}_{[r,Q]}, \boldsymbol{z}_r^k|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}) d\boldsymbol{z}_r^k \tag{5}$$

$$= \int_{\boldsymbol{z}_r^k} \prod_{q \in Q} \left(p_{\mathcal{G}}(x_{[r,q]}|\boldsymbol{z}_r^k, \boldsymbol{\theta}^k) \right) p_{\mathcal{G}}(\boldsymbol{z}_r^k|\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}) d\boldsymbol{z}_r^k \approx \frac{1}{J} \sum_{j=1}^{J} \prod_{q \in Q} p_{\mathcal{G}}(x_{[r,q]}|\boldsymbol{z}_r^{k,j}, \boldsymbol{\theta}^k), \tag{6}$$

where $\boldsymbol{z}_r^{k,j} \sim^{\mathcal{G}} |\{\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}\}$. Eq (5) suggests that `simulate` can be implemented by sampling $(\boldsymbol{x}_{[r,Q]}, \boldsymbol{z}_r^k) \sim^{\mathcal{G}} |\{\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}\}$ from the joint local posterior, then returning elements $\boldsymbol{x}_{[r,Q]}$. Eq (6) shows that `logpdf` can be implemented by first sampling the member latents $\boldsymbol{z}_r^k \sim^{\mathcal{G}} |\{\boldsymbol{x}_{[r,E]}, \boldsymbol{\theta}^k, \mathcal{D}\}$ from the local posterior; using the conditional independence constraint (2), the query $\boldsymbol{x}_{[r,Q]}$ then factors into a product of density terms for each element $x_{[r,q]}$.

To aggregate over $\{\boldsymbol{\theta}^k\}_{k=1}^K$, for `simulate` the runtime obtains the queried sample by first drawing $k \sim$ CATEGORICAL$(\{w^1, \dots, w^K\})$, then returns the sample $\boldsymbol{x}_{[r,Q]}$ drawn from trace $\boldsymbol{\theta}^k$. Similarly, `logpdf` is computed using the weighted Monte Carlo estimator (6). Algorithms 2a and 2b summarize implementations of `simulate` and `logpdf` in a general probabilistic programming environment.

Algorithm 2a `simulate` for CGPMs in a probabilistic programming environment.

```
1: function SIMULATE($\mathcal{G}$, $r$, $Q$, $\boldsymbol{x}_{[r,E]}$, $\boldsymbol{y}_r$)
2:     for $k = 1, \dots, K$ do                                          ▷ for each trace $k$
3:         if $\boldsymbol{z}_r^k \notin \boldsymbol{Z}^k$ then                        ▷ if member $r$ has unknown local latents
4:             $\boldsymbol{z}_r^k \sim^{\mathcal{G}} |\{\boldsymbol{\theta}^k, \boldsymbol{Z}^k, \mathcal{D}\}$                 ▷ sample them from the prior
5:         $w^k \leftarrow \prod_{e \in E} p_{\mathcal{G}}(x_{[r,e]} | \boldsymbol{\theta}^k, \boldsymbol{z}_r^k)$       ▷ weight the trace by likelihood of evidence
6:     $k \sim$ CATEGORICAL $(\{w^1, \dots, w^k\})$                     ▷ importance resample the traces
7:     $\{\boldsymbol{x}_{[r,Q]}, \boldsymbol{z}_r^k\} \sim^{\mathcal{G}} |\{\boldsymbol{\theta}^k, \boldsymbol{Z}^k, \mathcal{D} \cup \{\boldsymbol{y}_r, \boldsymbol{x}_{[r,E]}\}\}$   ▷ run a transition operator leaving target invariant
8:     return $\boldsymbol{x}_{[r,Q]}$                          ▷ select query variables from the resampled trace
```

Algorithm 2b `logpdf` for CGPMs in a probabilistic programming environment.

```
1: function LOGPDF($\mathcal{G}$, $r$, $\boldsymbol{x}_{[r,Q]}$, $\boldsymbol{x}_{[r,E]}$, $\boldsymbol{y}_r$)
2:     for $k = 1, \dots, K$ do                                          ▷ for each trace $k$
3:         Run steps 2 through 5 from Algorithm 2a                      ▷ retrieve the trace weight
4:         for $j = 1, \dots, J$ do                  ▷ obtain $J$ samples of latents in scope of member $r$
5:             $\boldsymbol{z}_r^{k,j} \sim^{\mathcal{G}} |\{\boldsymbol{\theta}^k, \boldsymbol{Z}^k, \mathcal{D} \cup \{\boldsymbol{y}_r, \boldsymbol{x}_{[r,E]}\}\}$  ▷ run a transition operator leaving target invariant
6:             $h^{k,j} \leftarrow \prod_{q \in Q} p_{\mathcal{G}}(x_{[r,q]} | \boldsymbol{\theta}^k, \boldsymbol{z}_r^{k,j})$             ▷ compute the density estimate
7:         $r^k \leftarrow \frac{1}{J} \sum_{j=1}^J h^{k,j}$         ▷ aggregate density estimates by simple Monte Carlo
8:         $q^k \leftarrow r^k w^k$                                      ▷ importance weight the estimate
9:     return $\log\left(\sum_{k=1}^K q^k\right) - \log\left(\sum_{k=1}^K w^k\right)$      ▷ weighted importance sampling over all traces
```

2.3 Inference in a composite network of CGPMs

This section shows how CGPMs are composed by applying the output of one to the input of another. This allows us to build complex probabilistic models out of simpler primitives directly as software. Section 3 demonstrates surface-level syntaxes in the Metamodeling Language for constructing these composite structures. We report experiments including up to three layers of composed CGPMs.

Let $\mathcal{G}^a$ be a CGPM with output $\boldsymbol{x}_*^a$ and input $\boldsymbol{y}_*^a$, and $\mathcal{G}^b$ have output $\boldsymbol{x}_*^b$ and input $\boldsymbol{y}_*^b$ (the symbol $*$ indexes all members $r \in \mathbb{N}$). The composition $\mathcal{G}_{\mathcal{B}}^b \circ \mathcal{G}_{\mathcal{A}}^a$ applies the subset of outputs $\boldsymbol{x}_{[*,\mathcal{A}]}^a$ of $\mathcal{G}^a$ to the inputs $\boldsymbol{y}_{[*,\mathcal{B}]}^b$ of $\mathcal{G}^b$, where $|\mathcal{A}| = |\mathcal{B}|$ and the variables are type-matched (Figure 1). This operation results in a new CGPM $\mathcal{G}^c$ with output $\boldsymbol{x}_*^a \cup \boldsymbol{x}_*^b$ and input $\boldsymbol{y}_*^a \cup \boldsymbol{y}_{[*,\backslash\mathcal{B}]}^b$. In general, a collection $\{\mathcal{G}^k : k \in [K]\}$ of CGPMs can be organized into a generalized directed graph $\mathcal{G}^{[K]}$, which itself is a CGPM. Node k is an "internal" CGPM $\mathcal{G}^k$, and the labeled edge $a_{\mathcal{A}} \to b_{\mathcal{B}}$ denotes the composition $\mathcal{G}_{\mathcal{A}}^a \circ \mathcal{G}_{\mathcal{B}}^b$. The directed acyclic edge structure applies only to edges between elements of different CGPMs in the network; elements $x_{[*,i]}^k, x_{[*,j]}^k$ within $\mathcal{G}^k$ may satisfy the more general constraint (1).

Algorithms 3a and 3b show sampling-importance-resampling and ratio-likelihood weighting algorithms that combine `simulate` and `logpdf` from each individual $\mathcal{G}^k$ to compute queries against network $\mathcal{G}^{[K]}$. The symbol $\pi^k = \{(p,t) : x_{[*,t]}^p \in \boldsymbol{y}_*^k\}$ refers to the set of all output elements from upstream CGPMs connected to the inputs of $\mathcal{G}^k$, so that $\{\pi^k : k \in [K]\}$ encodes the graph adjacency matrix. Subroutine 3c generates a full realization of all unconstrained variables, and weights forward samples from the network by the likelihood of constraints. Algorithm 3b is based on ratio-likelihood weighting (both terms in line 6 are computed by unnormalized importance sampling) and admits an analysis with known error bounds when `logpdf` and `simulate` of each $\mathcal{G}^k$ are exact [7].

Algorithm 3a `simulate` in a directed acyclic network of CGPMs.

```
1: function SIMULATE($\mathcal{G}^k$, $r$, $Q^k$, $\boldsymbol{x}_{[r,E^k]}^k$, $\boldsymbol{y}_r^k$, for $k \in [K]$)
2:     for $j = 1, \dots, J$ do                                 ▷ generate $J$ importance samples
3:         $(\boldsymbol{s}^j, w^j) \leftarrow$ WEIGHTED-SAMPLE $(\{\boldsymbol{x}_{[r,E^k]}^k : k \in [K]\})$      ▷ retrieve $j$th weighted sample
4:     $m \leftarrow$ CATEGORICAL $(\{w^1, \dots, w^J\})$                  ▷ resample by importance weights
5:     return $\{\boldsymbol{x}_{[r,Q^k]}^k \in \boldsymbol{s}_m : k \in [K]\}$      ▷ return query variables from the selected sample
```

Algorithm 3b `logpdf` in a directed acyclic network of CGPMs.

```
1: function SIMULATE($\mathcal{G}^k, r, \boldsymbol{x}^k_Q, \boldsymbol{x}^k_{[r,E^k]}, \boldsymbol{y}^k_r$, for $k \in [K]$)
2:     for $j = 1, \dots, J$ do                                              ▷ generate $J$ importance samples
3:         $(\boldsymbol{s}^j, w^j) \leftarrow$ WEIGHTED-SAMPLE $(\{\boldsymbol{x}^k_{[r,Q^k \cup E^k]} : k \in [K]\})$      ▷ joint density of query/evidence
4:     for $j = 1, \dots, J'$ do                                             ▷ generate $J'$ importance samples
5:         $(\boldsymbol{s}'^j, w'^j) \leftarrow$ WEIGHTED-SAMPLE $(\{\boldsymbol{x}^k_{[r,E^k]} : k \in [K]\})$             ▷ marginal density of evidence
6:     return $\log\left(\sum_{[J]} w^j / \sum_{[J']} w'^j\right) - \log(J/J')$          ▷ return likelihood ratio importance estimate
```

Algorithm 3c Weighted forward sampling in a directed acyclic network of CGPMs.

```
1: function WEIGHTED-SAMPLE (constraints: $\boldsymbol{x}^k_{[r,C^k]}$, for $k \in [K]$)
2:     $(\boldsymbol{s}, \log w) \leftarrow (\varnothing, 0)$                                   ▷ initialize empty sample with zero weight
3:     for $k \in$ TOPOSORT $(\{\pi^1, \dots, \pi^K\})$ do                ▷ topologically sort CGPMs using adjacency matrix
4:         $\tilde{\boldsymbol{y}}^k_r \leftarrow \boldsymbol{y}^k_r \cup \{x^p_{[r,t]} \in \boldsymbol{s} : (p,t) \in \pi^k\}$                ▷ retrieve required inputs at node $k$
5:         $\log w \leftarrow \log w +$ logpdf $(\mathcal{G}^k, r, \boldsymbol{x}^k_{[r,C^k]}, \varnothing, \tilde{\boldsymbol{y}}^k_r)$        ▷ update weight by likelihood of constraint
6:         $\boldsymbol{x}^k_{[r,\backslash C^k]} \leftarrow$ simulate $(\mathcal{G}^k, r, \backslash C^k, \boldsymbol{x}^k_{[r,C^k]}, \tilde{\boldsymbol{y}}^k_r)$           ▷ simulate unconstrained nodes
7:         $\boldsymbol{s} \leftarrow \boldsymbol{s} \cup \boldsymbol{x}^k_{[r,C^k \cup \backslash C^k]}$                           ▷ append all node values to sample
8:     return $(\boldsymbol{s}, w)$                                          ▷ return the overall sample and its weight
```

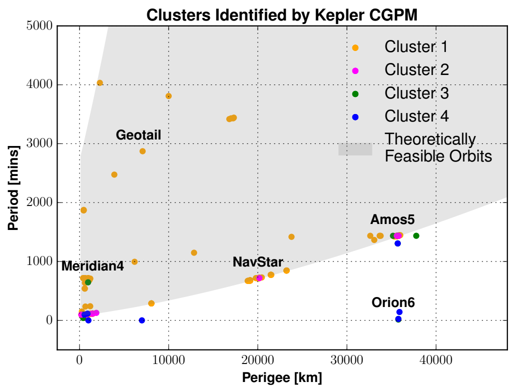
3 Analyzing satellites using CGPMs built from causal probabilistic programs, discriminative machine learning, and Bayesian non-parametrics

This section outlines a case study applying CGPMs to a database of 1163 satellites maintained by the Union of Concerned Scientists [12]. The dataset contains 23 numerical and categorical features of each satellite such as its material, functional, physical, orbital and economic characteristics. The list of variables and examples of three representative satellites are shown in Table 1. A detailed study of this database using BayesDB provided in [10]. Here, we compose the baseline CGPM in BayesDB, CrossCat [9], a non-parametric Bayesian structure learner for high dimensional data tables, with several CGPMs: a classical physics model written in VentureScript, a random forest classifier, factor analysis, and an ordinary least squares regressor. These composite models allow us to identify satellites that probably violate their orbital mechanics (Figure 2), as well as accurately infer the anticipated lifetimes of new satellites (Figure 3). We refer to [14, Section 6] for several more experiments on a broader set of data analysis tasks, as well as comparisons to baseline machine learning solutions.

Name	International Space Station	AAUSat-3	Advanced Orion 5 (NRO L-32, USA 223)
Country of Operator	Multinational	Denmark	USA
Operator Owner	NASA/Multinational	Aalborg University	National Reconnaissance Office (NRO)
Users	Government	Civil	Military
Purpose	Scientific Research	Technology Development	Electronic Surveillance
Class of Orbit	LEO	LEO	GEO
Type of Orbit	Intermediate	NaN	NaN
Perigee km	401	770	35500
Apogee km	422	787	35500
Eccentricity	0.00155	0.00119	0
Period minutes	92.8	100.42	NaN
Launch Mass kg	NaN	0.8	5000
Dry Mass kg	NaN	NaN	NaN
Power watts	NaN	NaN	NaN
Date of Launch	36119	41330	40503
Anticipated Lifetime	30	1	NaN
Contractor	Boeing Satellite Systems/Multinational	Aalborg University	National Reconnaissance Laboratory
Country of Contractor	Multinational	Denmark	USA
Launch Site	Baikonur Cosmodrome	Satish Dhawan Space Center	Cape Canaveral
Launch Vehicle	Proton	PSLV	Delta 4 Heavy
Source Used for Orbital Data	www.satellitedebris.net 12/12	SC - ASCR	SC - ASCR
longitude radians of geo	NaN	NaN	1.761037215
Inclination radians	0.9005899	1.721418241	0

Table 1: Variables in the satellite population, and three representative satellites. The records are multivariate, heterogeneously typed, and contain arbitrary patterns of missing data.

```
 1  CREATE TABLE satellites_ucs FROM 'satellites.csv';
 2  CREATE POPULATION satellites FOR satellites_ucs WITH SCHEMA ( GUESS STATTYPES FOR (*) );
 3
 4  CREATE METAMODEL satellites_hybrid FOR satellites WITH BASELINE CROSSCAT (
 5
 6    OVERRIDE GENERATIVE MODEL FOR type_of_orbit
 7    GIVEN apogee_km, perigee_km, period_minutes, users, class_of_orbit
 8    USING RANDOM_FOREST (num_categories = 7);
 9
10    OVERRIDE GENERATIVE MODEL FOR launch_mass_kg, dry_mass_kg, power_watts, perigee_km, apogee_km
11    USING FACTOR_ANALYSIS (dimensionality = 2);
12
13    OVERRIDE GENERATIVE MODEL FOR period_minutes
14    AND EXPOSE kepler_cluster_id CATEGORICAL, kepler_noise NUMERICAL
15    GIVEN apogee_km, perigee_km USING VENTURESCRIPT (program = '
16      define dpmm_kepler = () -> {                 // Definition of DPMM Kepler model program.
17        assume keplers_law = (apogee, perigee) -> {
18          (GM, earth_radius) = (398600, 6378);
19          a = .5*(abs(apogee) + abs(perigee)) + earth_radius;
20          2 * pi * sqrt(a**3 / GM) / 60 };
21        // Latent variable priors.
22        assume crp_alpha = gamma(1,1);
23        assume cluster_id_sampler = make_crp(crp_alpha);
24        assume noise_sampler = mem((cluster) -> make_nig_normal(1, 1, 1, 1));
25        // Simulator for latent variables (kepler_cluster_id and kepler_noise).
26        assume sim_cluster_id = mem((rowid, apogee, perigee) -> {
27          cluster_id_sampler() #rowid:1 });
28        assume sim_noise = mem((rowid, apogee, perigee) -> {
29          cluster_id = sim_cluster_id(rowid, apogee, perigee);
30          noise_sampler(cluster_id)() #rowid:2 });
31        // Simulator for observable variable (period_minutes).
32        assume sim_period = mem((rowid, apogee, perigee) -> {
33          keplers_law(apogee, perigee) + sim_noise(rowid, apogee, perigee) });
34        assume outputs = [sim_period, sim_cluster_id, sim_noise];     // List of output variables.
35      };
36      // Procedures for observing the output variables.
37      define obs_cluster_id = (rowid, apogee, perigee, value, label) -> {
38        $label: observe sim_cluster_id( $rowid, $apogee, $perigee) = atom(value); };
39      define obs_noise = (rowid, apogee, perigee, value, label) -> {
40        $label: observe sim_noise( $rowid, $apogee, $perigee) = value; };
41      define obs_period = (rowid, apogee, perigee, value, label) -> {
42        theoretical_period = run(sample keplers_law($apogee, $perigee));
43        obs_noise( rowid, apogee, perigee, value - theoretical_period, label); };
44      define observers = [obs_period, obs_cluster_id, obs_noise];     // List of observer procedures.
45      define inputs = ["apogee", "perigee"];                          // List of input variables.
46      define transition = (N) -> { default_markov_chain(N) };         // Transition operator.
47  '));
48  INITIALIZE 10 MODELS FOR satellites_hybrid;
49  ANALYZE satellites_hybrid FOR 100 ITERATIONS;
50  INFER name, apogee_km, perigee_km, period_minutes, kepler_cluster_id, kepler_noise FROM satellites;
```



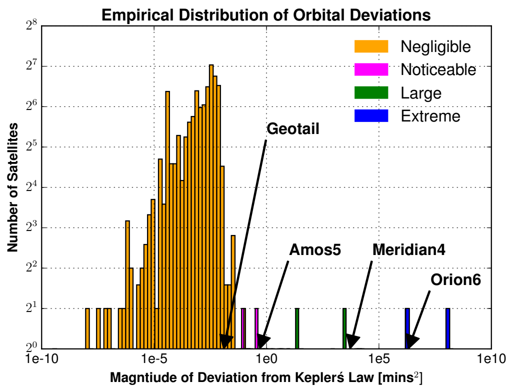


Figure 2: A session in BayesDB to detect satellites whose orbits are likely violations of Kepler's Third Law using a causal composable generative population model written in VentureScript. The `dpmm_kepler` CGPM (line 17) learns a DPMM on the residuals of each satellite's deviation from its theoretical orbit. Both the cluster identity and inferred noise are exposed latent variables (line 14). Each dot in the scatter plot (left) is a satellite in the population, and its color represents the latent cluster assignment learned by `dpmm_kepler`. The histogram (right) shows that each of the four detected clusters roughly translates to a qualitative description of the deviation: yellow (negligible), magenta (noticeable), green (large), and blue (extreme).

```
CREATE TABLE data_train FROM 'sat_train.csv';
.nullify data_train 'NaN';

CREATE POPULATION satellites FOR data_train
  WITH SCHEMA(
    GUESS STATTYPES FOR (*)
);

CREATE METAMODEL crosscat_ols FOR satellites
  WITH BASELINE CROSSCAT(
    OVERRIDE GENERATIVE MODEL FOR
        anticipated_lifetime
    GIVEN
        type_of_orbit, perigee_km, apogee_km,
        period_minutes, date_of_launch,
        launch_mass_kg
    USING LINEAR_REGRESSION
);

INITIALIZE 4 MODELS FOR crosscat_ols;
ANALYZE crosscat_ols FOR 100 ITERATION WAIT;

CREATE TABLE data_test FROM 'sat_test.csv';
.nullify data_test 'NaN';
.sql INSERT INTO data_train
    SELECT * FROM data_test;

CREATE TABLE predicted_lifetime AS
    INFER EXPLICIT
        PREDICT anticipated_lifetime
        CONFIDENCE prediction_confidence
    FROM satellites WHERE _rowid_ > 1000;
```

(a) Full session in BayesDB which loads the training and test sets, creates a hybrid CGPM, and runs the regression using CrossCat+OLS.

```
def dummy_code_categoricals(frame, maximum=10):

    def dummy_code_categoricals(series):
        categories = pd.get_dummies(
            series, dummy_na=1)
        if len(categories.columns) > maximum-1:
            return None
        if sum(categories[np.nan]) == 0:
            del categories[np.nan]
        categories.drop(
            categories.columns[-1], axis=1,
            inplace=1)
        return categories

def append_frames(base, right):
    for col in right.columns:
        base[col] = pd.DataFrame(right[col])

numerical = frame.select_dtypes([float])
categorical = frame.select_dtypes([object])

categorical_coded = filter(
    lambda s: s is not None,
    [dummy_code_categoricals(categorical[c])
        for c in categorical.columns])

joined = numerical

for sub_frame in categorical_coded:
    append_frames(joined, sub_frame)

return joined
```

(b) Ad-hoc Python routine (used by baselines) for coding nominal predictors in a dataframe with missing values and mixed data types.

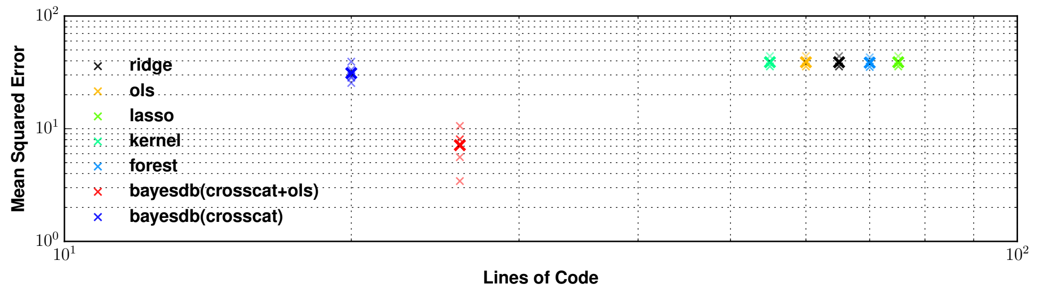


Figure 3: In a high-dimensional regression problem with mixed data types and missing data, the composite CGPM improves prediction accuracy over purely generative and purely discriminative baselines. The task is to infer the anticipated lifetime of a held-out satellite given categorical and numerical features such as type of orbit, launch mass, and orbital period. As feature vectors in the test set have missing entries, purely discriminative models (ridge, lasso, OLS) either heuristically impute missing features, or ignore the features and predict the anticipated lifetime using the mean in the training set. The purely generative model (CrossCat) can impute missing features from their joint distribution, but only indirectly mediates dependencies between the predictors and response through latent variables. The composite CGPM (CrossCat+OLS) in panel (a) combines advantages of both approaches; statistical imputation followed by regression on the features leads to improved predictive accuracy. The reduced code size is a result of using SQL, BQL, & MML, for preprocessing, model-building and predictive querying, as opposed to collections of ad-hoc scripts such as panel (b).

Figure 2 shows the MML program for constructing the hybrid CGPM on the satellites population. In terms of the compositional formalism from Section 2.3, the CrossCat CGPM (specified by the MML `BASELINE` keyword) learns the joint distribution of variables at the "root" of the network (i.e., all variables from Table 1 which do not appear as arguments to an MML `OVERRIDE` command). The `dpmm_kepler` CGPM in line 16 of the top panel in Figure 2 accepts `apogee_km` and `perigee_km` as input variables $\boldsymbol{y} = (A, P)$, and produces as output the `period_minutes` $\boldsymbol{x} = (T)$. These variables characterize the elliptical orbit of a satellite and are constrained by the relationships $e = (A - P)/(A + P)$ and $T = 2\pi\sqrt{((A + P)/2))^3/GM}$ where e is the `eccentricity` and GM

is a physical constant. The program specifies a stochastic version of Kepler's Law using a Dirichlet process mixture model for the distribution over errors (between the theoretical and observed period),

$$P \sim \text{DP}(\alpha, \textsc{Normal-Inverse-Gamma}(m, V, a, b)), \qquad (\mu_r, \sigma_r^2)|P \sim P$$
$$\epsilon_r|\{\mu_r, \sigma_r^2, \boldsymbol{y}_r\} \sim \textsc{Normal}(\cdot|\mu_r, \sigma_r^2), \text{ where } \epsilon_r := T_r - \textsc{Kepler}(A_r, P_r).$$

The lower panels of Figure 2 illustrate how the `dpmm_kepler` CGPM clusters satellites based on the magnitude of the deviation from their theoretical orbits; the variables (deviation, cluster identity, etc) in these figures are obtained from the BQL query on line 50. For instance, the satellite `Orion6` shown in the right panel of Figure 2, belongs to a component with "extreme" deviation. Further investigation reveals that `Orion6` has a recorded period 23.94 minutes, most likely a data entry error for the true period of 24 *hours* (1440 minutes); we have reported such errors to the maintainers of the database.

The data analysis task in Figure 3 is to infer the `anticipated_lifetime` x_r of a new satellite, given a set of features $\boldsymbol{y}_r$ such as its `type_of_orbit` and `perigee_km`. A simple OLS regressor with normal errors is used for the response $p_{\mathcal{G}^{\text{ols}}}(x_r|\boldsymbol{y}_r)$. The CrossCat baseline learns a joint generative model for the covariates $p_{\mathcal{G}^{\text{crosscat}}}(\boldsymbol{y}_r)$. The composite CGPM `crosscat_ols` built Figure 3 (left panel) thus carries the full joint distribution over the predictors and response $p_{\mathcal{G}}(\boldsymbol{x}_r, \boldsymbol{y}_r)$, leading to more accurate predictions. Advantages of this hybrid approach are further discussed in the figure.

4 Related Work and Discussion

This paper has shown that it is possible to use a computational formalism in probabilistic programming to uniformly apply, combine, and compare a broad class of probabilistic data analysis techniques. By integrating CGPMs into BayesDB [10] and expressing their compositions in the Metamodeling Language, we have shown it is possible to combine CGPMs synthesized by automatic model discovery [9] with custom probabilistic programs, which accept and produce multivariate inputs and outputs, into coherent joint probabilistic models. Advantages of this hybrid approach to modeling and inference include combining the strengths of both generative and discriminative techniques, as well as savings in code complexity from the uniformity of the CGPM interface.

While our experiments have constructed CGPMs using VentureScript and Python implementations, the general probabilistic programming interface of CGPMs makes it possible for BayesDB to interact with a variety systems such as BUGS [15], Stan [1], BLOG [11], Figaro [13], and others. Each of these systems provides varying levels of model expressiveness and inference capabilities, and can be used to be construct domain-specific CGPMs with different performance properties based on the data analysis task on hand. Moreover, by expressing the data analysis tasks in BayesDB using the model-independent Bayesian Query Language [10, Section 3], CGPMs can be queried without necessarily exposing their internal structures to end users. Taken together, these characteristics help illustrate the broad utility of the BayesDB probabilistic programming platform and architecture [14, Section 5], which in principle can be used to create and query novel combinations of black-box machine learning, statistical modeling, computer simulation, and probabilistic generative models.

Our applications have so far focused on CGPMs for analyzing populations from standard multivariate statistics. A promising area for future work is extending the computational abstraction of CGPMs, as well as the Metamodeling and Bayesian Query Languages, to cover analysis tasks in other domains such longitudinal populations [3], statistical relational settings [6], or natural language processing and computer vision. Another extension, important in practice, is developing alternative compositional algorithms for querying CGPMs (Section 2.3). The importance sampling strategy used for compositional `simulate` and `logpdf` may only be feasible when the networks are shallow and the constituent CGPMs are fairly noisy; better Monte Carlo strategies or perhaps even variational strategies may be needed for deeper networks. Additional future work for composite CGPMs include (i) algorithms for jointly learning the internal parameters of each individual CGPM, using, e.g., imputations from its parents, and (ii) new meta-algorithms for structure learning among a collection of compatible CGPMs, in a similar spirit to the non-parametric divide-and-conquer method from [9].

We hope the formalisms in this paper lead to practical, unifying tools for data analysis that integrate these ideas, and provide abstractions that enable the probabilistic programming community to collaboratively explore these research directions.

References

[1] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *J Stat Softw*, 2016.

[2] G. Casella and R. Berger. *Statistical Inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002.

[3] M. Davidian and D. M. Giltinan. *Nonlinear models for repeated measurement data*, volume 62. CRC press, 1995.

[4] L. Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pages 260–265. ACM, 1986.

[5] D. Fink. A compendium of conjugate priors. 1997.

[6] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1300–1309, 1999.

[7] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[8] V. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014.

[9] 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.

[10] 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.

[11] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. 1 blog: Probabilistic models with unknown objects. *Statistical relational learning*, page 373, 2007.

[12] U. of Concerned Scientists. UCS Satellite Database, 2015.

[13] A. Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137, 2009.

[14] F. Saad and V. Mansinghka. Probabilistic data analysis with probabilistic programming. *arXiv preprint arXiv:1608.05347*, 2016.

[15] D. J. Spiegelhalter, A. Thomas, N. G. Best, W. Gilks, and D. Lunn. Bugs: Bayesian inference using gibbs sampling. *Version 0.5,(version ii) http://www. mrc-bsu. cam. ac. uk/bugs*, 19, 1996.

BayesDB: A probabilistic programming system for querying the probable implications of data

Vikash Mansinghka vkm@mit.edu
Richard Tibbetts tibbetts@mit.edu
Jay Baxter jbaxter@mit.edu
Computer Science & Artificial Intelligence Laboratory
Department of Brain & Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Pat Shafto p.shafto@louisville.edu
Baxter Eaves b0eave01@louisville.edu
Department of Psychology
University of Louisville
Louisville, KY 40292, USA

Abstract

Is it possible to make statistical inference broadly accessible to non-statisticians without sacrificing mathematical rigor or inference quality? This paper describes BayesDB, a probabilistic programming platform that aims to enable users to query the probable implications of their data as directly as SQL databases enable them to query the data itself. This paper focuses on four aspects of BayesDB: (i) BQL, an SQL-like query language for Bayesian data analysis, that answers queries by averaging over an implicit space of probabilistic models; (ii) techniques for implementing BQL using a broad class of multivariate probabilistic models; (iii) a semi-parametric Bayesian model-builder that auomatically builds ensembles of factorial mixture models to serve as baselines; and (iv) MML, a "meta-modeling" language for imposing qualitative constraints on the model-builder and combining baseline models with custom algorithmic and statistical models that can be implemented in external software. BayesDB is illustrated using three applications: cleaning and exploring a public database of Earth satellites; assessing the evidence for temporal dependence between macroeconomic indicators; and analyzing a salary survey.

Keywords: Probabilistic programming, Bayesian inference, probabilistic databases, multivariate statistics, nonparametric Bayes, automatic machine learning

Acknowledgements: VKM would like to thank Alexey Radul, Feras Saad, and Taylor Campbell for helpful discussions and contributions to a prototype implementation. This research was supported by DARPA (under the XDATA and PPAML programs), IARPA (under research contract 2015-15061000003), the Office of Naval Research (under research contract N000141310333), the Army Research Office (under agreement number W911NF-13-1-0212), the Bill & Melinda Gates Foundation, and gifts from Analog Devices and Google.

1. Introduction

Is it possible to make statistical inference broadly accessible to non-statisticians without sacrificing mathematical rigor or inference quality? This paper describes BayesDB, a system that enables users to query the probable implications of their data as directly as SQL databases enable them to query the data itself. By combining ordinary SQL with three new primitives — `SIMULATE`, `INFER`, and `ESTIMATE` — users of BayesDB can detect predictive relationships between variables, retrieve statistically similar data items, identify anomalous data points and variables, infer missing values, and synthesize hypothetical subpopulations. The default modeling assumptions that BayesDB makes are suitable for a broad class of problems (Mansinghka et al., 2015; Wasserman, 2011), but statisticians can customize these assumptions when necessary. BayesDB also enables domain experts that lack statistical expertise to perform qualitative model checking (Gelman et al., 1995) and encode simple forms of qualitative prior knowledge.

BayesDB consists of four components, integrated into a single probabilistic programming system:

1. The Bayesian Query Language (BQL), an SQL-like query language for Bayesian data analysis. BQL programs can solve a broad class of data analysis problems using statistically rigorous formulations of cleaning, exploration, confirmatory analysis, and predictive modeling. BQL defines these primitive operations for these workflows in terms of Bayesian model averaging over results from an implicit set of multivariate probabilistic models.

2. A mathematical interface that enables a broad class of multivariate probabilistic models, called generative population models, to be used to implement BQL. According to this interface, a data generating process defined over a fixed set of variables is represented by (i) an infinite array of random realizations of the process, including any observed data, and (ii) algorithms for simulating from arbitrary conditional distributions and calculating arbitrary conditional densities. This interface permits many statistical operations to be implemented once, independent of the specific models that will be used to apply these operations in the context of a particular data table.

3. The BayesDB Meta-modeling Language (MML), a minimal probabilistic programming language. MML includes constructs that enable statisticians to integrate custom statistical models — including arbitrary algorithmic models contained in external software — with the output of a broad class of Bayesian model building techniques. MML also includes constructs for specifying qualitative dependence and independence constraints.

4. A hierarchical, semi-parametric Bayesian "meta-model" that automatically builds ensembles of generalized mixture models from database tables. These ensembles serve as baseline data generators that BQL can use for data cleaning, initial exploration, and other routine applications.

This design insulates end users from most statistical considerations. Queries are posed in a qualitative probabilistic programming language for Bayesian data analysis that hides the details of probabilistic modeling and inference. Baseline models can be built automatically and customized by statisticians when necessary. All models can be critically assessed and qualitatively validated via predictive checks that compare synthetic rows (generated via BQL's `SIMULATE` operation) with rows from the original data. Instead of hypothesis testing, dependencies between variables are obtained via Bayesian model selection.

BayesDB is "Bayesian" in two ways:

1. In BQL, the objects of inference are rows, and the underlying probability model forms a "prior" probability distribution on the fields of these rows. This is then constrained by row-specific observations to create a posterior distribution of field values. Without this prior, it would be impossible to simulate rows or infer missing values from partial observations.

2. In MML, the default meta-model is Bayesian in that it assigns a prior probability to a very broad class of probabilistic models and narrows down on probable models via Bayesian inference. This prior is unusual in that it encodes a state of ignorance rather than a strong inductive constraint. MML also provides instructions for augmenting this prior to incorporate qualitative and quantitative domain knowledge.

In practice, it is useful to use BQL for Bayesian queries against models built using non-Bayesian or only partially Bayesian techniques. For example, MML supports composing the default meta-model with modeling techniques specified in external code that need not be Bayesian. However, the default is to be Bayesian for both model building and query interpretation, as this ensures the broadest applicability of the results.

This paper focuses on the technical details of BQL, the data generator interface, the meta-model, and the MML. It also illustrates the capabilities of BayesDB using three applications: cleaning and exploring a public database of Earth satellites, discovering relationships in measurements of macroeconomic development of countries, and analyzing salary survey data. Empirical results are based on a prototype implementation that embeds BQL into `sqlite3`, a lightweight, open-source, in-memory database.

1.1 A conceptual illustration

This section illustrates data analysis using the MML and BQL on a synthetic example based on analysis of electronic health records. SQL databases make it easy to load data from disk and run queries that filter and retrieve the contents. The first step in using BayesDB is to load data that describes a statistical (sub)population into a table, with one row per member of the population, and one column per variable:

```
CREATE POPULATION patients WITH DATA FROM patients.csv;

SELECT age, has_heart_disease FROM patients WHERE age > 30 LIMIT 3;
```

```
 age | has_heart_disease
-----+------------------
  66 |       ???
  44 |       yes
  31 |       ???
```

Once data has been loaded, a *population schema* needs to be specified. This schema specifies the statistical characteristics of each example. For example, whether it is categorical or numerical, and if it is categorical, how many outcomes are there and how is each outcome represented. After an initial schema has been specified — using a mix of automatic inference and manual specification — the schema can be customized using instructions in the Meta-modeling Language (MML).

```
GUESS POPULATION SCHEMA FOR patients;

ALTER POPULATION SCHEMA FOR patients
SET DATATYPE FOR num_hosp_visits TO COUNT;


CREATE DEFAULT METAMODEL FOR patients;
ALTER METAMODEL FOR patients ENSURE will_readmit DEPENDENT ON dialysis;

ALTER METAMODEL FOR patients
MODEL infarction GIVEN gender, age, weight, height, cholesterol, bp
USING CUSTOM MODEL FROM infarction_regression.py;
```

One distinctive feature of MML is that it includes instructions for *qualitative probabilistic programming*. These instructions control the behavior of the automatic modeling machinery in the MML runtime. In this example, these constraints include the assertion of a dependence between the presence of a chronic kidney condition and future hospital readmissions. They also include the specification of a custom statistical model for the `infarction` variable, illustrating one way that discriminative and non-probabilistic approaches to inference can be integrated into BayesDB.

The next step is to use the MML to build an ensemble of general-purpose models for the data, subject to the specified constraints:

```
INITIALIZE 100 MODELS FOR patients;
ANALYZE patients FOR 3 HOURS CHECKPOINT EVERY 10 MINUTES;
```

Each of these 100 models is a *generative population model* (GPM) that represents the joint distribution on all possible measurements of an infinite population with the given population schema. These models are initially drawn accordingt to a broad prior probability distribution over a large hypothesis space of possible GPMs. Until the observed data has been analyzed, BQL will thus report broad uncertainty for all its query responses.

Once the models are sufficiently adapted to the data, it is possible to query its probable implications. The following query quantifies over columns, rather than rows, and retrieves the probability of a marginal dependence between three (arbitrary) variables and `height`:

```
ESTIMATE COLUMN NAME, PROBABILITY OF DEPENDENCE WITH height
FROM COLUMNS OF patients LIMIT 3;
```

column name	*p(dep. with height)*
height	*1.0*
infarction	*0.08*
gender	*0.99*

Point predictions can be accessed by using the `INFER` instruction, a natural generalization of `SELECT` from SQL:

```
INFER age, has_heart_disease FROM patients
WHERE age > 30 WITH CONFIDENCE 0.8 LIMIT 3;
```

age	*has_heart_disease*
66	yes
44	*yes*
31	*???*

In this example, only one of the missing values could be inferred with the specified confidence level. The probabilistic semantics of `CONFIDENCE` will be discussed later in this paper.

BQL also makes it straightforward to generate synthetic sub-populations subject to a broad class of constraints:

```
SIMULATE height, weight, blood_pressure FROM patients 3 TIMES
GIVEN gender = male AND age < 10
```

height	*weight*	*blood_pressure*
46	*80*	*110*
38	*60*	*80*
39	*119*	*120*

The `SIMULATE` operator gives BQL users access to samples from the posterior predictive distribution induced by the implicit underlying set of models. This is directly useful for predictive modeling and also decision-theoretic choice implemented using Monte Carlo estimation of expected utility (Russell and Norvig, 2003). It also enables predictive checking: samples from `SIMULATE` can be compared to the results returned by `SELECT`. Finally, domain experts can use `SIMULATE` to scrutinize the implications of the underlying model ensemble, both quantitatively and qualitatively.

2. Example Analyses

This section describes three applications of the current BayesDB prototype:

1. Exploring and cleaning a public database of Earth satellites.
2. Assessing the evidence for dependencies between indicators of global poverty
3. Analyzing data from a salary survey.

BQL and MML constructs are introduced via real-world uses; a discussion of their formal interpretation is provided in later sections.

2.1 Exploring and cleaning a public database of Earth satellites

The Union of Concerned Scientists maintains a database of 1000 Earth satellites. For the majority of satellites, it includes kinematic, material, electrical, political, functional, and economic characteristics, such as dry mass, launch date, orbit type, country of operator, and purpose. Here we show a sequence of interactions with a snapshot of this database using the `bayeslite` implementation of BayesDB.

2.1.1 Inspecting the data.

We start by loading the data and looking at a sample. This process uses a combination of ordinary SQL and convenience functions built into `bayeslite`. The first step is to create a population from the raw data:

```
CREATE POPULATION satellites FROM ucs_database.csv
```

One natural query is to find the International Space Station, a well-known satellite:

```
SELECT * FROM satellites WHERE Name LIKE 'International Space Station%'
```

Variable	**Value**
Name	International Space Station (ISS ...)
Country_of_Operator	Multinational
Operator_Owner	NASA/Multinational
Users	Government
Purpose	Scientific Research
Class_of_Orbit	LEO
Type_of_Orbit	Intermediate
Perigee_km	401
Apogee_km	422
Eccentricity	0.00155
Period_minutes	92.8
Launch_Mass_kg	NaN
Dry_Mass_kg	NaN
Power_watts	NaN
Date_of_Launch	36119
Anticipated_Lifetime	30
Contractor	Boeing Satellite Systems (prime)/Multinational
Country_of_Contractor	Multinational
Launch_Site	Baikonur Cosmodrome
Launch_Vehicle	Proton
Source_Used_for_Orbital_Data	www.satellitedebris.net 12/12
longitude_radians_of_geo	NaN
Inclination_radians	0.9005899

This result row illustrates typical characteristics of real-world databases such as heterogeneous data types and missing values.

2.1.2 Building baseline models.

Before exploring the implications of the data, it is necessary to obtain a collection of probabilistic models. The next two MML instructions produce a collection of 16 models, using roughly 4 minutes of analysis total.

```
INITIALIZE 16 MODELS FOR satellites;
ANALYZE satellites FOR 4 MINUTES WAIT;
```

Each of the 16 models is a separate GPM produced by an independent Markov chain for approximate posterior sampling in the default semi-parametric factorial mixture meta-model described earlier. This number of models and amount of computation is typical for the exploratory analyses done with our prototype implementation; this is sufficient for roughly 100 full sweeps of all latent variables.

2.1.3 Answering hypotheticals.

The satellites database should in principle inform the answers to a broad class of hypothetical or "what if?" questions. For example, consider the following question:

> *Suppose you receive a report indicating the presence of a previously undetected satellite in geosynchronous orbit with a dry mass of 500 kilograms. What countries are most likely to have launched it, and what are its likely purposes?*

Answering this question requires knowledge of satellite engineering, orbital mechanics, and the geopolitics of the satellite industry. It is straightforward to answer this question using BQL. The key step is to generate a synthetic population of satellites that reflect the given constraints:

```
SIMULATE country_of_operator, purpose FROM satellites
GIVEN Class_of_orbit = GEO, Dry_mass_kg = 500 LIMIT 1000;
```

Figure 1 shows the results of a simple aggregation of these results, counting the marginal frequencies of various countries and purposes and sorting accordingly. The most probable explanation, carrying roughly 25% of the probability mass, is that it is a communications satellite launched by the USA. It is also plausible that it might have been launched other major space powers such as Russia or China, and that it might have a military purpose.

The satellites data are too sparse and ambiguous for frequency counting to be a viable alternative. Consider an approach based on finding satellites that match the discrete `GEO` constraint and are within some ad-hoc tolerance around the observed dry mass:

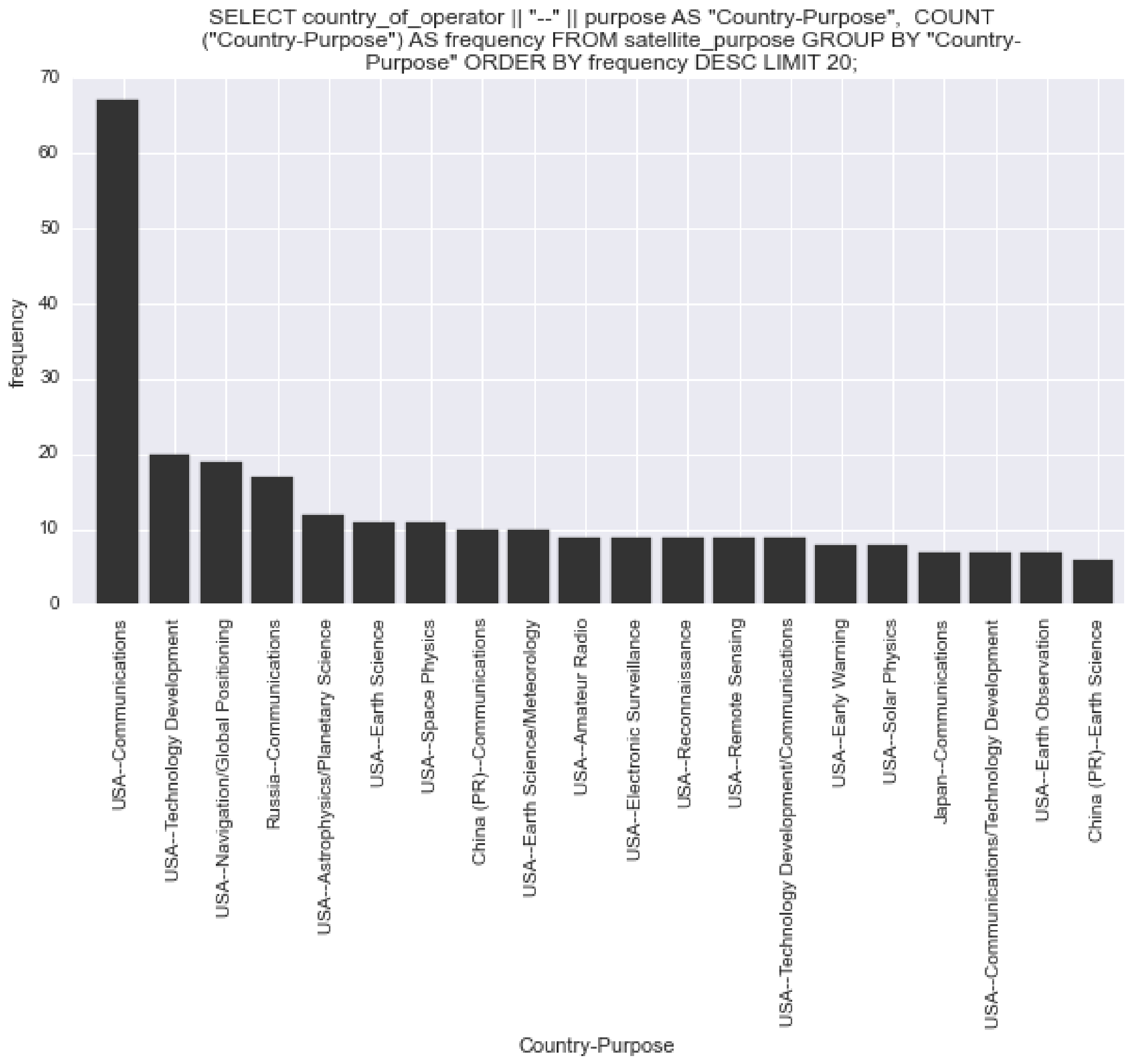
```
SELECT country_of_operator, purpose, Class_of_orbit, Dry_mass_kg
FROM satellites
WHERE Class_of_orbit = "GEO" AND Dry_Mass_kg BETWEEN 400 AND 600;
```

This SQL query returns just 2 satellites, both Indian:

	Country_of_Operator	**Purpose**	**Class_of_Orbit**	**Dry_Mass_kg**
0	India	Communications	GEO	559
1	India	Meteorology	GEO	500

Presuming our intuition about satellite mass is flawed, we might issue another query to look at a broader range of satellites:

```
SELECT country_of_operator, purpose, Class_of_orbit, Dry_mass_kg
FROM satellites
WHERE Class_of_orbit = 'GEO' AND Dry_Mass_kg BETWEEN 300 AND 700
```



```
SELECT country_of_operator || "--" || purpose AS "Country-Purpose",  COUNT
("Country-Purpose") AS frequency FROM satellite_purpose GROUP BY "Country-
Purpose" ORDER BY frequency DESC LIMIT 20;
```

Figure 1: **The most probable countries and purposes of a satellite with a 500 kilogram dry mass in geosynchronous orbit.** See main text for discussion.

The results still do not give any real insight into the likely purpose of this satellite:

	Country_of_Operator	Purpose	Class_of_Orbit	Dry_Mass_kg
0	Malaysia	Communications	GEO	650
1	Israel	Communications	GEO	646
2	Luxembourg	Communications	GEO	700
3	Russia	Communications	GEO	620
4	China (PR)	Earth Science	GEO	620
5	China (PR)	Earth Science	GEO	620
6	China (PR)	Earth Science	GEO	620
7	India	Communications	GEO	559
8	India	Navigation	GEO	614
9	India	Meteorology	GEO	500
10	Malaysia	Communications	GEO	650
11	Multinational	Earth Science/Meteorology	GEO	320
12	United Kingdom	Communications	GEO	660
13	Norway	Communications	GEO	646

Without deep expertise in satellites, and significant expertise in statistics, it is difficult to know whether or not these results can be trusted. How does the set of satellites vary as the thresholds on `Dry_Mass_kg` are adjusted? How locally representative and comprehensive is the coverage afforded by the data? Are there indirect, multivariate dependencies that ought to be taken into account, to determine which satellites are most similar? How should existing satellites be weighted to make an appropriate weighted sample against which to calculate frequencies? In fact, small modifications to the tolerance on `Dry_Mass_kg` yield large changes in the result set.

2.1.4 Identifying predictive relationships between variables.

A key exploratory task is to identify those variables in the database that appear to predict one another. This is closely related to the key confirmatory analysis question of assessing the evidence for a predictive relationship between any two particular variables.

To quantify the evidence for (or against) a predictive relationship between two pairs of variables, BQL relies on information theory. The notion of dependence between two variables A and B is taken to be mutual information; the amount of evidence for dependence is then the probability that the mutual information between A and B is nonzero. If the population models are obtained by posterior inference in a meta-model — as is the case with MML — then this probability approximates the posterior probability (or strength of evidence) that the mutual information is nonzero.

```
ESTIMATE DEPENDENCE PROBABILITY FROM PAIRWISE COLUMNS OF satellites;
```

Figure 2 shows the results from this query. There are several groups of variables with high probability of mutual interdependence. For example, we see a definite block of geopolitically related variables, such as the country of contractor & operator, the contractor's identity, and the location of the satellite (if it is in geosynchronous orbit). The kinematic variables describing orbits, such as perigee, apogee, period, and orbit class, are also shown as strongly interdependent. A domain expert with sufficiently confident domain knowledge can use this overview of the predictive relationships to assess the value of the data and the validity of the baseline models.

It is also instructive to compare the heatmap of pairwise dependence probabilities with alternatives from statistics. Figure 2 also shows heatmap that results from datatype-appropriate measures of correlation. The results from correlation are sufficiently noisy that it would be difficult to trust inferences from techniques that use correlation to select variables. Furthermore, the most causally unambiguous relationships, such as the kinematic constraints relating perigee, apogee, and orbital period, not detected by correlation.

2.1.5 Detecting multivariate anomalies.

Another key aspect of exploratory analysis is identifying anomalous values, including both (univariate) outliers and multivariate anomalies. Anomalies can arise due to errors in data acquisition, bugs in upstream preprocessing software (including binning of continuous variables or translating between different discrete outcomes), and runtime failures. Anomalies can also arise due to genuine surprises or changes in the external environment.

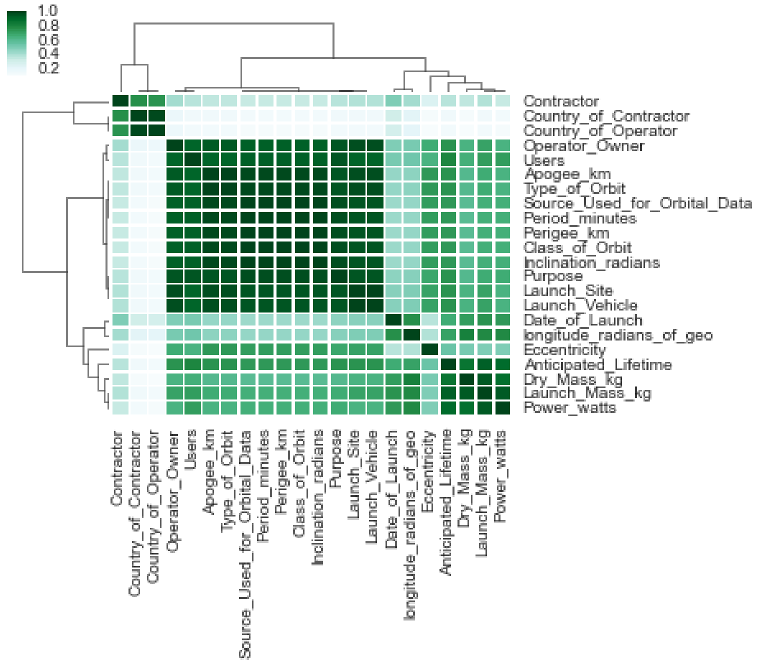
Using BQL, multivariate anomalies can be detected by assessing the predictive probability density of each measurement, and ordering from least to most probable. Here we illustrate this using a simple example: ordering the geosynchronous satellites according to the probability of their recorded orbital period:

```
ESTIMATE name, class_of_orbit, period_minutes AS TAU,
PREDICTIVE PROBABILITY OF period_minutes AS "Pr[TAU]"
FROM satellites
ORDER BY ‘‘Pr[TAU]’’ ASCENDING LIMIT 10
```

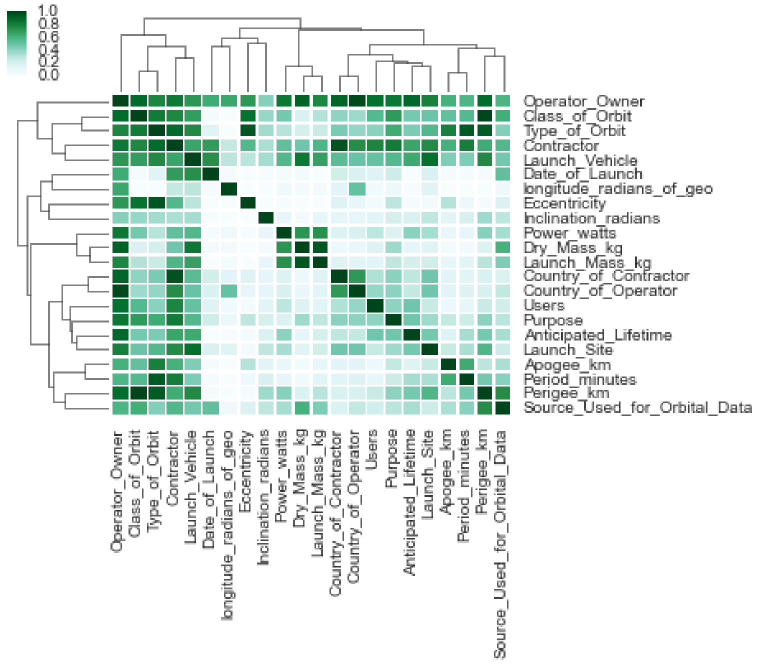
This BQL query produces the following table of results:

	Name	**Class_of_Orbit**	**TAU**	**Pr[TAU]**
0	AEHF-3 (Advanced Extremely High Frequency sate...	GEO	1306.29	0.001279
1	AEHF-2 (Advanced Extremely High Frequency sate...	GEO	1306.29	0.001292
2	DSP 20 (USA 149) (Defense Support Program)	GEO	142.08	0.002656
3	Intelsat 903	GEO	1436.16	0.003239
4	BSAT-3B	GEO	1365.61	0.003440
5	Intelsat 902	GEO	1436.10	0.003492
6	SDS III-6 (Satellite Data System) NRO L-27, Gr...	GEO	14.36	0.003811
7	Advanced Orion 6 (NRO L-15, USA 237)	GEO	23.94	0.003938
8	SDS III-7 (Satellite Data System) NRO L-38, Dr...	GEO	23.94	0.003938
9	QZS-1 (Quazi-Zenith Satellite System, Michibiki)	GEO	1436.00	0.004446

Recall that a geosynchronous orbit should take 24 hours or 1440 minutes. Rows 7 and 8 appear to be unit conversion errors (hours rather than minutes). Rows 2 and 6 appear to be decimal placement errors. Note that row 2 is not an outlier: some satellites have an orbital



(a) A heatmap depicting the pairwise probabilities of dependence between all pairs of variables. The rows and columns are both permuted according to a single ordering obtained via agglomerative hierarchical clustering to highlight multivariate interactions.



(b) The pairwise correlation matrix; note that many causal relationships are not detected by simple correlations. See main text for discussion.

Figure 2: **Detecting predictive relationships in the satellites database.**

period of roughly two hours. It is only anomalous in the context of the other variables that are probably predictive of orbital period, such as orbit class.

2.1.6 Inferring missing values.

A key application of predictive modeling is inferring point predictions for missing measurements. This can be necessary for cleaning data before downstream processing. It can also be of intrinsic interest, e.g. in classification problems. The satellites database has many missing values. Here we show an `INFER` query that infers missing orbit types and returns both a point estimate and the confidence in that point estimate:

```
INFER EXPLICIT
anticipated_lifetime, perigee_km, period_minutes, class_of_orbit,
PREDICT type_of_orbit AS inferred_orbit_type CONFIDENCE inferred_orbit_type_conf
FROM satellites
WHERE type_of_orbit IS NULL;
```

This form of `INFER` uses the `EXPLICIT` modifier that exposes both predicted values and their associated confidence levels to be included in the output. Figure 3 shows a visualization of the results. The panel on the bottom left shows that the confidence depends on the orbit class and on the predicted value for the inferred orbit type. For example, there is typically moderate to high confidence for the orbit type of `LEO` satellites — and high confidence (but some variability in confidence) for those with `Sun-Synchronous` orbits. Satellites with `Elliptical` orbits may be assigned a `Sun-Synchronous` type with moderate confidence, but for other target labels confidence is generally lower. After examining the overall distribution on confidences, it can be natural to filter `INFER` results based on a manually specified confidence threshold. Note that many standard techniques for imputation from statistics correspond to `INFER ... WITH CONFIDENCE 0`.

2.1.7 Integrating a kinematic model for elliptical orbits.

Can we improve over the baseline models by integrating causal knowledge about satellites? MML can be used to compose GPMs built by the default model builder with algorithmic and/or statistical models specified as external software. Here we integrate a simple model for elliptical orbits:

```
ALTER METAMODEL FOR satellites
MODEL perigee_km, apogee_km GIVEN period_minutes, eccentricity
USING CUSTOM MODEL FROM stochastic_kepler.py;
ANALYZE FOREIGN PREDICTORS FOR 1 MINUTE;
```

The underlying foreign predictor implements Kepler's laws:

$$\begin{aligned} R_{min} &= \tau^{\frac{2}{3}}(1.0 - \epsilon) - R_{GEO} \\ R_{max} &= \tau^{\frac{2}{3}}(1.0 + \epsilon) - R_{GEO} \\ X_{(r^*,\texttt{apogee\_km})} &\sim N(R_{min}, \sigma^2) \\ X_{(r^*,\texttt{perigee\_km})} &\sim N(R_{max}, \sigma^2) \end{aligned}$$

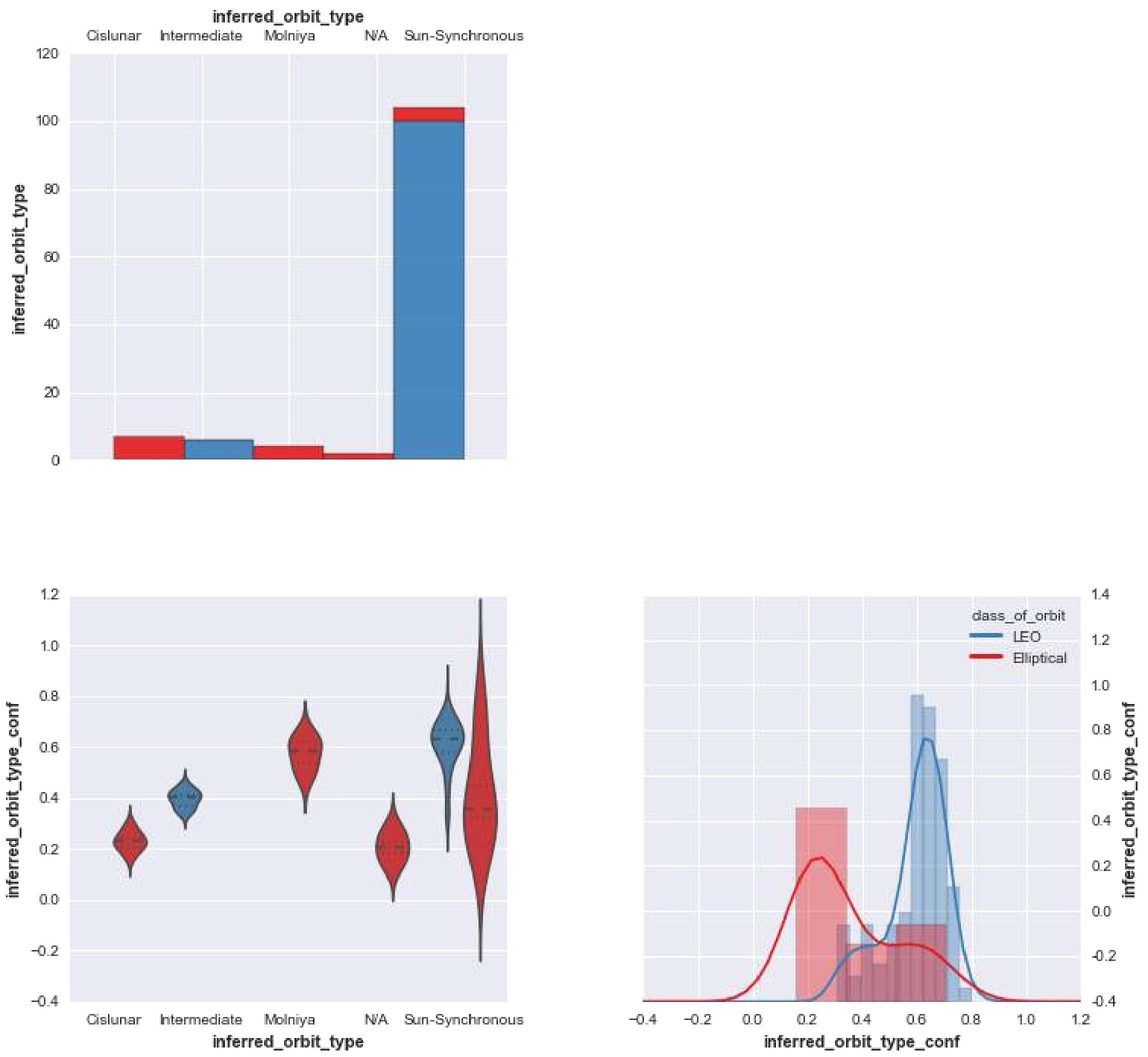


Figure 3: **A visualization of inferred point estimates for `type_of_orbit` and the confidence in those point estimates.** See main text for discussion.

Here, ϵ is set via the `eccentricity` measurement for row r^*, and τ is set via the `period_minutes` measurement. R_{GEO} is a fixed constant inside the foreign predictor representing the radius of the Earth in kilometers. σ is a parameter that determines the noise and is set when the variable subset for the foreign predictor instantiation is `ANALYZE`d. Note that foreign predictors are essentially GPMs and accordingly must implement generic simulate(...) and logpdf(...) methods. For algorithmic forward models with numerical outputs, MML provides a default wrapper that uses importance sampling with resampling to approximately generate conditional samples and estimate marginal densities. This is how Kepler's laws — in the form of a forward simulation — are turned into a generative model for kinematic variables that can be conditionally simulated in arbitrary directions.

Figure 4 and Figure 5 show the results. A detailed discussion of the relative merits of empirical versus analytical modeling is beyond the scope of this paper. However, it is clear that neither the empirical approach nor the analytical approach is universally dominant. The empirical approach is able to correctly locate the empirical probability mass — including multiple modes — but underfits. The orbital mechanics approach yields inferences that are typically in much tighter accord with the kinematic data. This is unsurprising: these are the patterns of covariation that led to the development of quantitative models and "hard" natural sciences. However, there are many satellites for which Kepler's laws are not in accord with the data. This reflects many factors, including data quality errors as well as legitimate gaps between idealized mathematical laws and fine-grained empirical records of real-world phenomena. For example, at the time Kepler's laws were formulated, orbiting bodies lacked engines.

2.1.8 Combing random forests, causal models, and nonparametric Bayes.

Because MML supports model composition, it is straightforward to build hybrid models that integrate techniques from subfields of machine learning that might seem to be in conflict. Figure 6 shows the transcript of a complete MML session that builds such a hybrid model. Random forests are used to classify orbits into types; Kepler's Laws are used to relate period, perigee, and apogee; and the default semi-parametric Bayesian meta-model is used for all remaining variables (with two variables coming with overridden datatypes).

Later sections of this paper explain how these three modeling approaches are combined to answer individual BQL queries.

2.2 Assessing the evidence for dependencies between indicators of global poverty

In the early 21st century it is widely believed that resolving extreme poverty around the world will be accomplished by empowering individuals to resolve their poverty. Governments and NGOs encourage this process through a variety of interventions, many of them combining material assistance with policy changes. In principle, policies should be driven by quantitative data-driven understanding of international economic development. In practice, international economic data is sparse, unreliable, and highly aggregated. These data limitations create substantial obstacles to understanding the situational context of successful or unsuccessful interventions and policies.

```
SIMULATE period_minutes, apogee_km FROM satellites_kepler LIMIT 100;
```

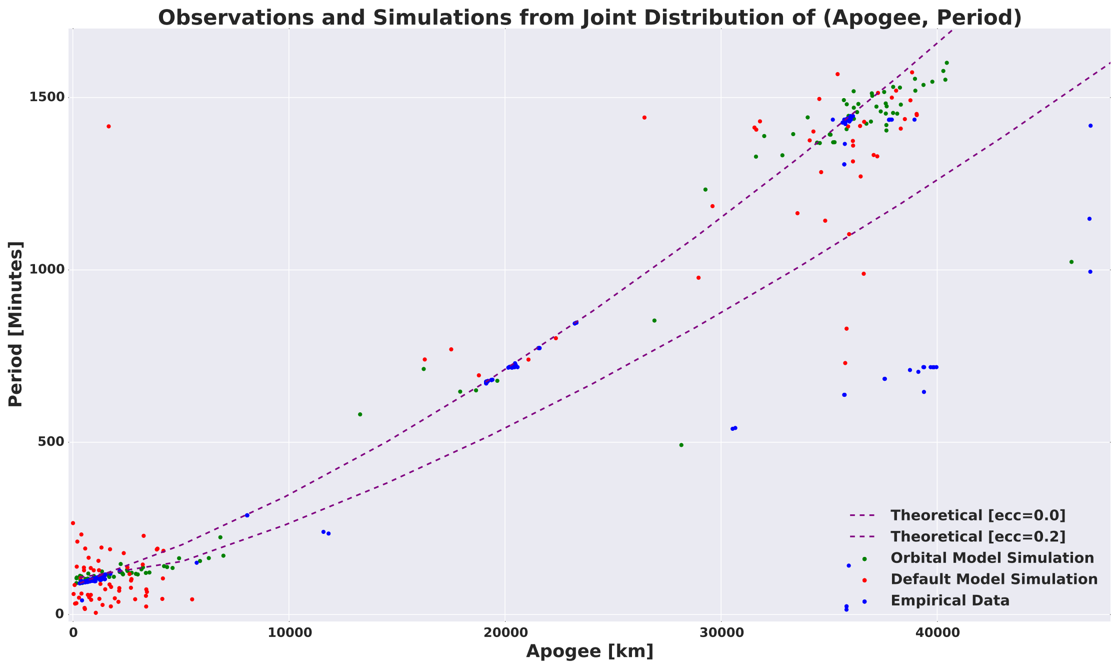
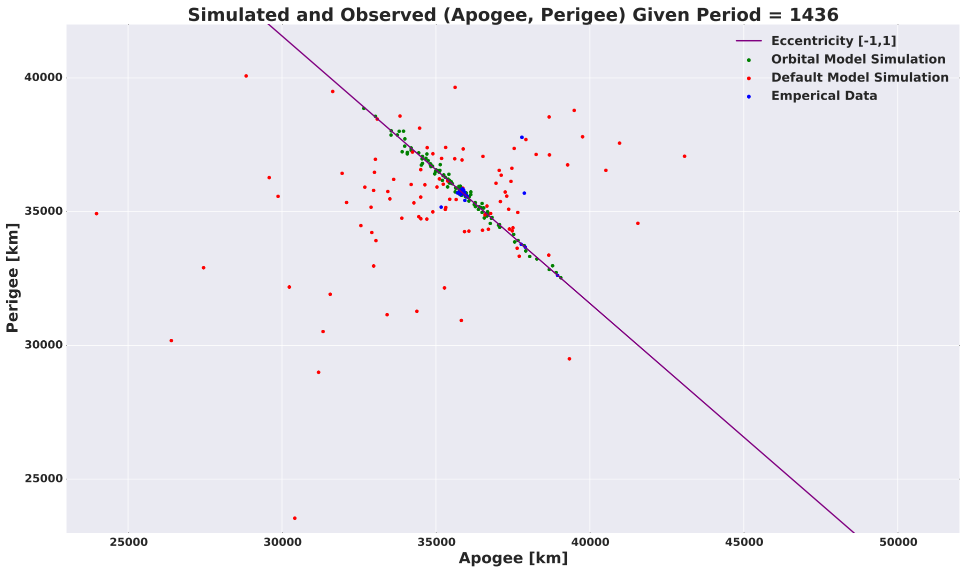


Figure 4: **Integrating empirical baseline models with a noisy version of Kepler's laws.** Neither the empirical approach nor the analytical approach is universally dominant in terms of accuracy. See main text for discussion.

```
SIMULATE perigee_km, apogee_km FROM satellites_kepler ASSUMING
                  period_minutes = 1436 LIMIT 100;
```



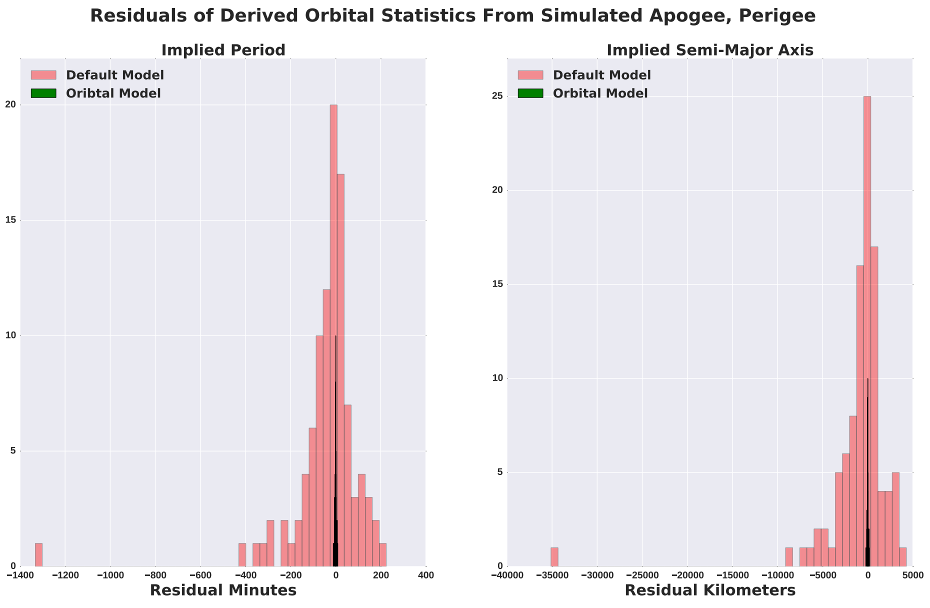


Figure 5: **Comparing conditional predictions of an empirical model with Kepler's laws.** See main text for discussion.

```
CREATE POPULATION satellites
  FROM ucs_satellites.csv

CREATE METAMODEL sat_keplers ON satellites
  USING composer(
  random_forest (
    Type_of_Orbit (CATEGORICAL)
      GIVEN Apogee_km, Perigee_km,
            Eccentricity, Period_minutes,
            Launch_Mass_kg, Power_watts,
            Anticipated_Lifetime,
 Class_of_orbit
  ),
  foreign_model (
    source = 'keplers_laws.py',
    Period_Minutes (NUMERICAL)
      GIVEN Perigee_km, Apogee_km
  ),
  default_metamodel (
    Country_of_Operator CATEGORICAL,
    Inclination_radians NUMERICAL
  )
);

INITIALIZE 16 MODELS FOR satellites;
ANALYZE satellites FOR 4 MINUTES;
```

Figure 6: A complete MML session that builds a hybrid model integrating techniques from subfields of machine learning that might seem to be in conflict.

The “Gapminder” data set, collected and curated by Hans Rosling at the Karolinska Institutet, is the most well known and extensive sets of longitudinal global developmental indicators. Representing over 500 indicators, 400 countries, and 500 years of data, it covers the colonial era, industrial revolution, socio-political upheavals around the world in the 20th century, and the first decade of the 21st. Containing over 2 million observations, the data has been used as the basis for a compelling set of data animations and the most widely viewed TED talk on statistics.

To date, analysis of this data has been minimal, as it requires intensive preprocessing and cleaning. Different analytical methdologies require different approaches to imputation and variable selection; as a result, results from different teams are difficult to compare. Here we show how to explore the data with BayesDB and assess the evidence for predictive relationships between different macroeconomic measures of development.

2.2.1 Exploring the Data with SQL

The raw form of the data is ∼500 Excel spreadsheets, each containing longitudinal data for ∼300 countries over ∼100 years. However, the dataset only contains ∼2 million observations, i.e. 97% of the data is missing. Figure 7 shows key indicators of the data around size, missing records, and the relationship between data availability and countries, records, and years. The primary data is mmodeled in SQL as a âĂIJfactâĂİ table structure. This relatively-normalized representation easily models the sparse matrix and allows us to use a combination of SQL and Python data science tools to craft our population structures.

The histograms in Figure 7 show the breadth and also the variability of the data. The histogram by year in Figure 7a shows that data is complete for only recent history, and in fact that some predicted data continues into the future, and that data for some indicators is only available every 10 years. The histogram by country in Figure 7a shows that data availability varies by country (the most described country is Sweden), that many countries have reasonably complete data, but that there is a long tail of countries with sparse data, including countries that no longer exist and with inconsistent or disputed naming. Figure 7c shows that there is also a variance by indicators, because different measurements are collected by different agencies with different expectations and data policies.

The data has already been subject to extensive visualization and descriptive analytics by the Gapminder project. This paper focuses on the use of MML to model the data and BQL to query its probable implications.

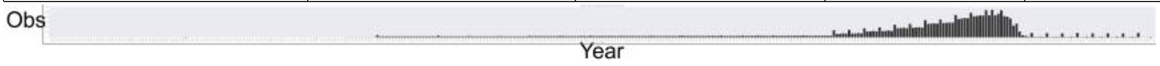
2.2.2 Detecting Basic and Longitudinal Dependence

Our analysis focuses on the 53 variables with most complete data for the years 1999-2008. It is straightforward to create an ensemble of models for this subset:

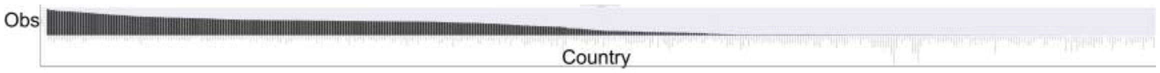
```
GUESS POPULATION SCHEMA FOR dense_gapminder;
INITIALIZE 64 MODELS FOR dense_gapminder;
ANALYZE todo FOR 300 MINUTES WAIT;
```

The probability of dependence heatmap that results is shown in Figure 8. Indicators such as total population and urban percentage form blocks containing their values for all 10 years contained in the dataset. This shows that the default GPM was able to extract the

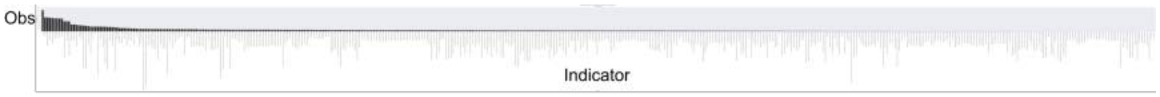
observation_count	indicator_count	country_count	year_count	coverage
2082193	464	405	364	0.03044



(a) **Observation volume by year, 1086-2100.** Note that the most complete data is for only recent history.



(b) **Observation volume by country.** Note that the data becomes very limited for some countries, and includes countries that no longer exist.



(c) **Observation volume by indicator.** Note that some indicators are much more complete or extensive in terms of year and country than others.

Figure 7: **Gapminder data volume measures.** The dataset contains longitudinal records of ∼500 macroeconomic variables for ∼300 countries, spanning a ∼100 year period. However, roughly 97% of the data is missing.

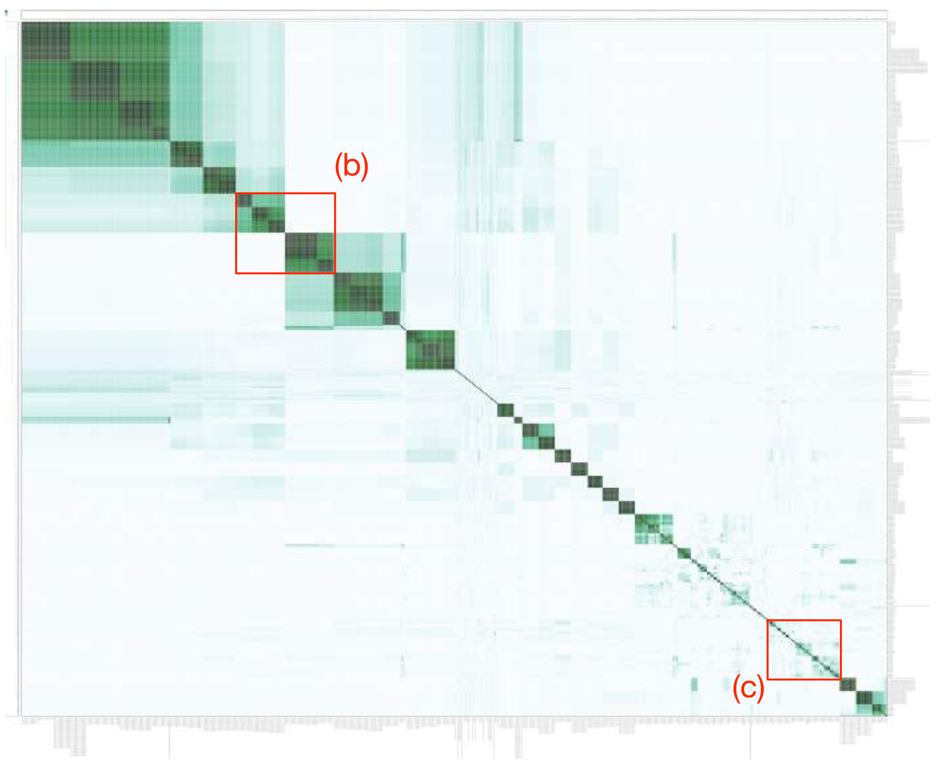
temporal dependence in these indicators. In other cases, such as measurements of the number of people killed in floods, the year to year dependence is much weaker. The heatmap also shows dependence between indicators, such as the block in the top right corner combining indicators of stress, urbanization, and fertility rate. Finally, it segregates data according to type sof indicators, as can be seen in Figure 8b where there is a sharp break from total measurements to per-capital.

If we analyze just the data for the year 2002, using 32 models for 3 minutes, we get a heatmap that highlights the dependence and independence between indicators. Figure 9 shows the details.

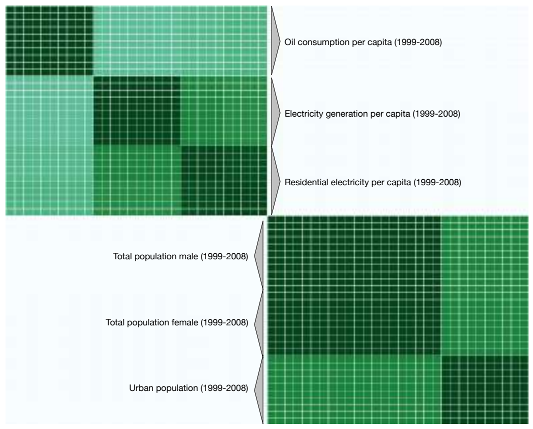
2.2.3 Measuring the Similarity of Countries

In order to help with the delivery of international aid and the design and analysis of interventions, decision makers often want a richer understanding of the similarities between countries. With BQL we can formulate these queries in general or against specific attributes. Figure 10 shows country similarities for different indicators. As expected, changing the indicator of interest can produce a very different similarity structure. Analyses that presume a single global similarity measure cannot pick up this context-specific structure.

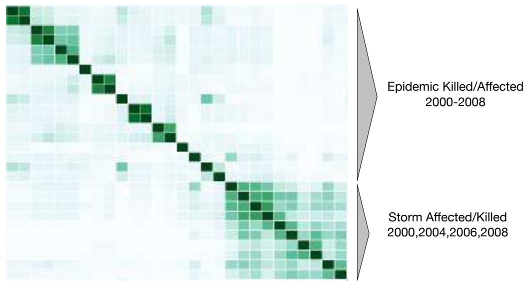
The authors are involved in an ongoing research partnership with the Bill and Melinda Gates Foundation aimed at integrating the Gapminder data with other relevant sources, including qualitative knowledge from domain eperts, and using it to drive empirically grounded policy and aid interventions.



(a) Probability of dependence heatmap for 40 indicators over 10 years.



(b) Indicators form 10 year runs, with a sharp break from totals to per capita.



(c) Natural disaster indicators cluster but do not have strong year-to-year dependence.

Figure 8: **Probability of dependence heatmap for the Gapminder data.** BayesDB detects temporal dependence within some indicators but not others, as well as dependence between some indicators (but not others). See main text for discussion.

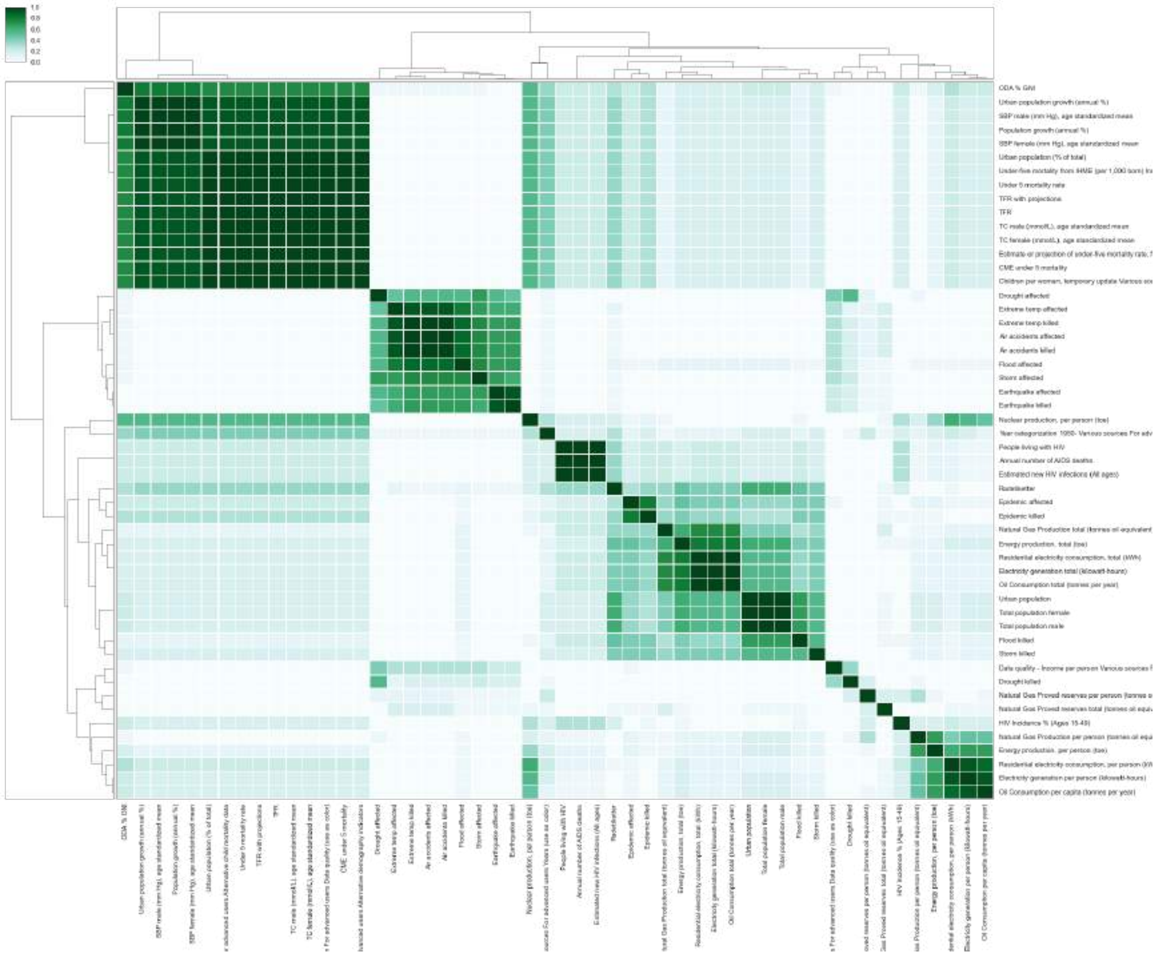
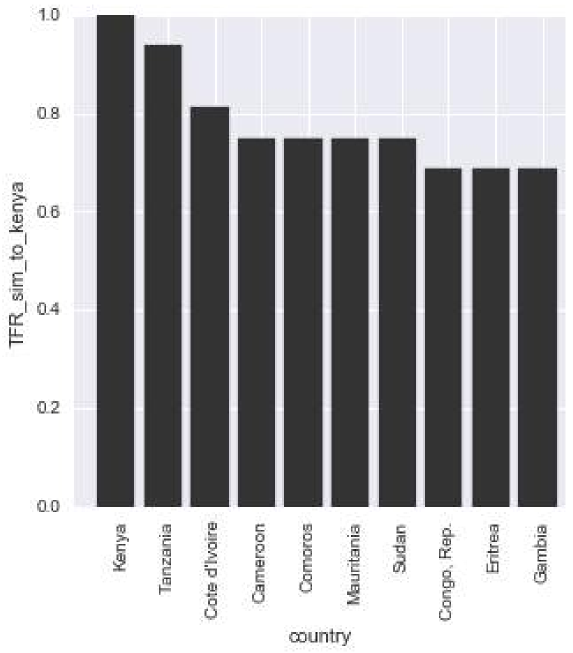
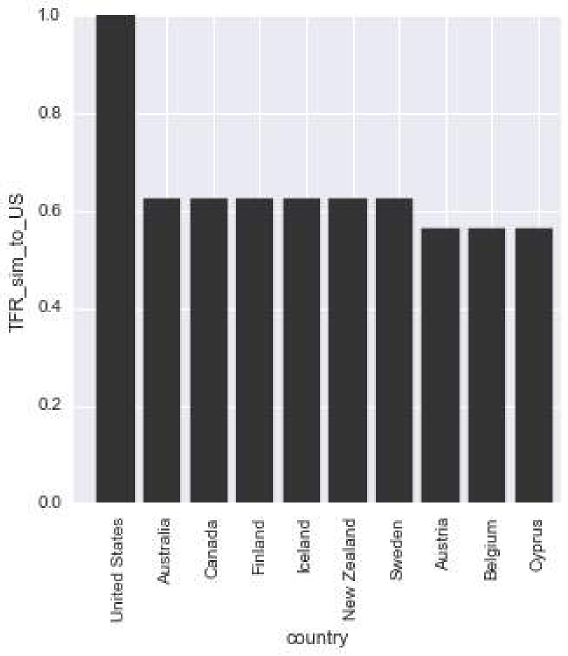


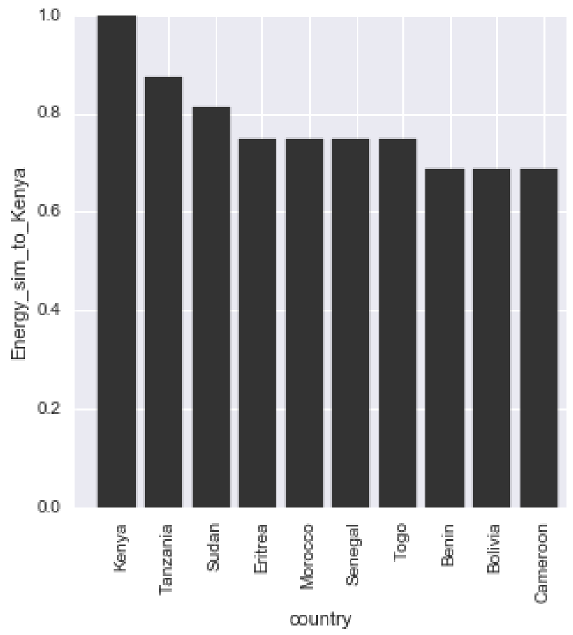
Figure 9: **Probability of dependence for 40 indicators in 2002.** Of particular note are the blocks for population growth rates, natural disasters, HIV, total energy usage, and per capita energy usage.



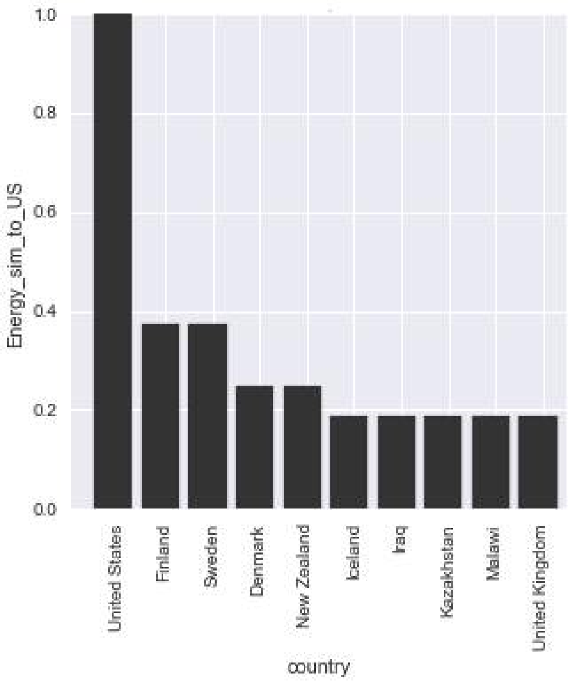
(a) Countries enumerated in decending order of similarity to Kenya on Total Fertility Rate.



(b) Countries enumerated in descending order of similarity to the United States on Total Fertility Rate.

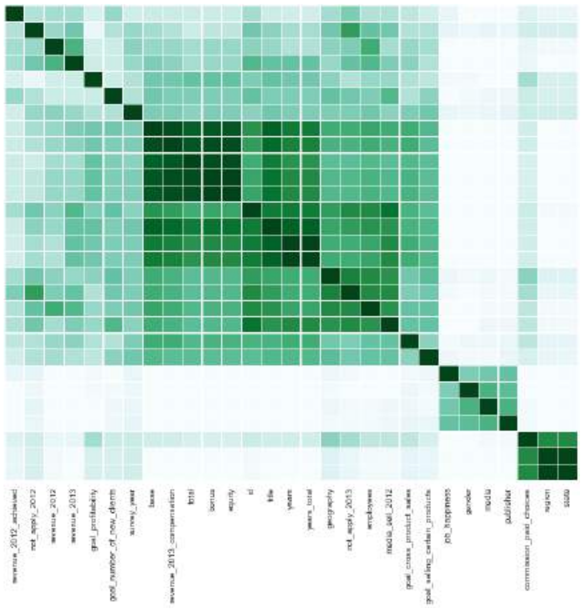


(c) Countries enumerated in decending order of similarity to Kenya on per capita energy production.

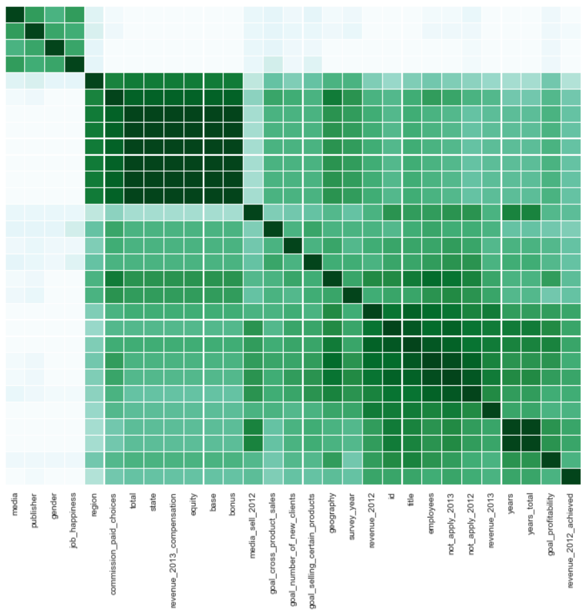


(d) Countries enumerated in descending order of similarity to the United States on per capita energy production.

Figure 10: **Examining the similarity of countries.** Kenya and the United States are similar to different countries, and the similarity structure with respect to per capita energy production and total fertility rate are significantly different.



(a) Probability of dependence between columns with default metamodel settings.



(b) Probability of dependence between columns when `state` is dependent on salary columns.

Figure 11: **Probability of dependence heat maps with and without dependence assertions in MML.** Note that in the second figure stronger dependencies were resolved overall.

2.3 Analyzing a salary survey

Surveys are a common source of multivariate data and a potentially appealing application for BayesDB. Here we show a preliminary analysis of a web-administered anonymous salary survey. Participants shared their compensation details along with information about their title, years of service, acheivements, employer, and geography.

2.4 Controlling Models with Qualitative Assumptions

This salary population provides an instructive example of applying qualitiative assumptions to a model. In this case, the first analysis of compensation data finds that geographic location (state, region) is not a factor in compensation. Domain experts suggest that is implausible, that cost of living and the competitive market in different cities is a significant factor in compensation of the survey participants. The following code can be used to apply this qualitative assumption:

```
ALTER METAMODEL FOR salary ENSURE total, equity, base, bonus DEPENDENT
ON state;
```

Without asserting the dependence, state is inferred to be dependent on region and independent of performance. After asserting a qualitative constraint, the probability of dependence heat map changes. Not only are the squares implied by that depencence colored to 1.0, but other columns have re-aligned in their modeling. In particular, given this assumption, there appears to be more evidence of dependence between the 2012 and 2013 measures and core indicators such as years in the job, bonus, the presence of an equity stake, etc. Also, there appears to be less evidence that job title impacts the key compensation variables.

3. The Bayesian Query Language

The Bayesian Query Language (BQL) formalizes Bayesian data analysis without exposing the end user to model parameters, priors, and posteriors. This section describes the statistical operations that are implemented by the core BQL instruction set. It also describes the modeling formalism that is used to implement BQL.

3.1 Generative Population Models

BQL programs are executed against a weighted collection of *generative population models* (GPMs). At present, GPMs can be built in two ways:

1. Specified directly as external software libraries.
2. Inferred from data via probabilistic inference in a meta-model written in BayesDB's Meta-modeling Language.

GPMs can respond to queries about the joint distribution of the underlying data generating process as a whole or about the predictive distribution for a specific member of the population. The population can be thought of as a table, where individual members are specified by row indexes.

Each GPM induces a random table with a finite number of columns and an infinite number of rows, where each cell contains a random variable. BQL treats each BayesDB generator as a model of the data generating process underlying its associated table of observations. It is sometimes useful to query a GPM about hypothetical members of the population. This can be performed by using a row whose index r^* may not be associated with any actual member; this can be guaranteed by generating a unique row index.

Each GPM is described by a *schema* $\mathcal{S}$ that must be compatible with the population schema for the population to which it is being applied. This schema is a tuple containing $(\texttt{typed-outputs}, \texttt{typed-inputs}, \texttt{body})$. The `typed-outputs` component specifies the column indexes and statistical types of each column that the data generator will be responsible for producing. The `typed-inputs` component specifies the indexes and statistical types of each column that the data generator can read from. The `body` is an opaque binary that contains any GPM-specific configuration information, such as a probabilistic program.

Mathematically, the internals of a GPM $\mathcal{G} = (\Theta, \mathcal{Z}, O)$ consists of three parts:

1. Measurement-specific latent variables $\mathcal{Z} = \cup z_{(r,c)}$.

   There may be overlap between the latent variables for different measurements. If a GPM cannot track dependencies internally — or if it is based on a model class in which all measurements are coupled — then $z_{(r,c)} = \mathcal{Z}$.

2. Population-level latent variables Θ.

   These are all latent variables that remain well-defined in the absence of all measurements. Examples include hyper-parameters and mixture component parameters.

3. Observations $O = \{(r_i, c_i, x_{(r_i,c_i)})\}$.

   These correspond to the observed measurements.

For example, a naive Bayesian GPM lacks any measurement-specific variables, i.e. $z_{(r,c)} = \emptyset$, and is completely characterized by a single vector of parameters $\Theta = \vec{\theta_c}$ for the probability models for each column. A finite mixture GPM would have $z_{(r,c)} = \{z_r\}$ be the cluster assignment for each row, and have $\Theta = \{\theta_{(c,l)} | l \in \mathcal{Z}\}$ be the component model parameters for each cluster.

Generative population models are required to satisfy the following conditional independence constraint:

$$x_{(r,c)}|\Theta, z_{(r,c)} \perp\!\!\!\perp x_{(r',c')}|\Theta, z_{(r',c')} \text{ unless } (r,c) = (r',c')$$

Note in particular that the observations O need not be conditioned on directly, given Θ and $z_{(r,c)}$. This formalizes the requirement that the dependencies between the measurements in the population are completely mediated by the population-level latent variables and all relevant measurement-specific latent variables. In general, no other independence constraints are enforced by the interface. GPMs can thus be built around dense, highly-coupled model families such as low-dimensional latent spaces and convolutional neural networks.

3.1.1 An interface to generative population models

A GPM must implement the following interface:

1. $\mathcal{G} = \{\Theta, \mathcal{Z}\}$ = initialize(schema = $\mathcal{S}$)

   Initialize a data generator with the given schema and return the resulting data generator $\mathcal{G}$. It ensures that storage has been allocated for the random variables Θ and $\mathcal{Z}$, storing the global latent variables and the local latent variables, respectively.

2. $\vec{s_i}$ = simulate($\mathcal{G}$, givens = $\{(r_j, c_j, x_{(r,c_j)})\}$, targets = $\{r_k, c_k\}$, N)

   Generate N sampled values $\{\vec{s_i}\}$ from the specified distribution:

   $$\{\vec{s_i}\} \sim \{X_{(r_k,c_k)}\}|\{X_{(r_j,c_j)} = x_{(r_j,c_j)}\}, \Theta, \{z_{r,c}|(r,c) \in \{r_j, c_j\} \cup \{r_k, c_k\}\}$$

   The set of valid distributions includes all finite-dimensional joint distributions obtainable by conditioning on the values of arbitrary measurements and marginalizing over another arbitrary set.

3. $\log p$ = logpdf($\mathcal{G}$, givens = $\{(r_j, c_j, x_{(r_j,c_j)})\}$, query = $\{(r_k, c_k, q_{(r,c_k)})\}$)

   Evaluate the log probability density of the specified conditional/marginal distribution at a target point:

   $$\log p = \log p(\{X_{(r_k,c_k)} = q_{(r_k,c_k)}\}|\{X_{(r_j,c_j)} = x_{(r_j,c_j)}\}, \Theta, \{z_{r,c}|(r,c) \in \{r_j, c_j\} \cup \{r_k, c_k\}\})$$

4. $d =$ kl-divergence-given-G($\mathcal{G}$, measurements_A $= \{(r_i, c_i^a)\}$, measurements_B $= \{(r_j, c_j^b)\}$, conditions_C $= \{(r_k, c_k^c, x^k)\}$)

   This estimates the KL divergence of the set of measurements A from the set of measurements B, conditioned on the given constraints C. KL calculations are central to model-independent data analysis. For example, to detect predictive relationships, it suffices to check for non-zero mutual information, which can be reduced to calculating the KL between the joint distribution over two variables and the product of the marginals.

   It is included in the GPM interface because that allows a GPM implementer to supply an optimized implementation. Where such an implementation is not available, the KL can be estimated via simple Monte Carlo estimation:

$$
\begin{aligned}
\mathrm{D}^{\mathcal{G}}_{KL}(\{X_{(r_i,c_i^a)}\}, \{X_{(r_j,c_j^b)}\}) &= \sum_{\{x_i\} \in dom(\{X_{(r_i,c_i^a)}\})} p\left(\{X_{(r_i,c_i^a)}\} = \{x_i\}\,|\mathcal{G}\right) log \left(\frac{p\left(\{X_{(r_j,c_j^b)}\} = \{x_i\}\,|\mathcal{G}\right)}{p\left(\{X_{(r_i,c_i^a)}\} = \{x_i\}\,|\mathcal{G}\right)} \right) \\
&\approx \sum_{\{x_i\}^k} log \left(\frac{p\left(\{X_{(r_j,c_j^b)}\} = \{x_i\}^k\,|\mathcal{G}\right)}{p\left(\{X_{(r_i,c_i^a)}\} = \{x_i\}^k)\,|\mathcal{G}\right)} \right) \\
&\text{with} \{x_i\}^k \sim \{X_{(r_i,c_i^a)}\}
\end{aligned}
$$

This interface is intentionally quite general. It needs to support an open set of primitives for Bayesian data analysis. This paper focuses on the subset of this interface where all measurements come from the same row. All the BQL operations used in this paper can be reduced to explicit invocations of *simulate*, *logpdf*, and to Monte Carlo estimates of Kullback-Leibler divergences implemented in terms them. Some GPMs can significantly optimize some of these operations relative to Monte Carlo baselines; such optimizations are likely to be important in practice but are beyond the scope of this paper.

3.1.2 Weighted collections of generative population models.

BQL is executed against a weighted collection of GPMs $\mathcal{M}$:

$$\mathcal{M} = \{(w_i, \mathcal{G}_i)\}$$

In principle, these collections can include GPMs drawn from different model classes. The weights are treated as prior probabilities. This paper focuses on the case where the GPMs come from a single meta-model, each produced by independent runs of a single Markov chain for posterior inference in the meta-model given all available measurements. In this case, assigning unit weights to all models $w_i = 1$ results in BQL queries based on a Monte Carlo approximation to Bayesian model averaging.

3.2 Core instructions: `SIMULATE`, `ESTIMATE`, and `INFER`

Data analysis workflows in BQL are built around three core classes of statistical operations:

1. Generating samples from predictive probability distributions, including both completions of existing rows in a data table as well as predictive distributions over hypothetical rows.
2. Estimating predictive probability densities and approximating derived information-theoretic quantities.
3. Summarizing multi-modal probability distributions with single values.

These capabilities are exposed via three basic extensions to SQL that each combine results from individual GPMs in different ways. They can be composed with ordinary SQL to solve a broad range of data analysis tasks:

1. *Detecting predictive relationships between variables*: `ESTIMATE COLUMN PROBABILITY OF DEPENDENCE WITH ...`

   This yields an estimate of the marginal probability of dependence between the specified columns. This is equivalent to the probability that the mutual information between those two variables is nonzero, integrating over the weighted collection of GPMs that BayesDB maintains. If the GPMs are produced by an asymptotically consistent estimator of the joint distribution, then these probabilities will reflect non-linear, heteroscedastic, or context-specific dependencies that statistical aggregates (such as correlation or linear regression coefficients) will not.

2. *Regression, classification, semi-supervised learning, and imputation*: `INFER ...`

   Each of these predictive modeling tasks requires filling in point estimates in different conditions. All of these can be viewed as special cases of `INFER`, which handles arbitrary patterns of missing values and both continuous and discrete prediction targets.

3. *Anomaly/outlier detection*: `ORDER BY PROBABILITY OF` *`col`* `ASCENDING LIMIT` *`k`*

   Anomalous cells can be found by predictive checking: identify the cells that are least likely under the inferred constellation of models. These may not be outliers in the standard univariate sense: the low probability may be due to interactions between several variables, even though each variable on its own is marginally typical.

4. *Retrieving similar rows*: `ORDER BY SIMILARITY TO` *`row`*

   A broad class of structured search operations can be performed via information-theoretic measures of similarity between rows. These are useful in both data exploration and in more targeted search.

5. *Predictive model checking*: `SIMULATE ...`

   By comparing aggregates from the output of `SIMULATE` to the output of the analogous `SELECT` statements, it is possible to do predictive checking without having to mention models, parameters, priors, or posteriors.

3.2.1 SIMULATE: generating samples from arbitrary predictive distributions.

The first, called SIMULATE, provides a flexible interface to sampling from posterior predictive distributions:

> SIMULATE *target columns* FROM *population* [WHERE *row filter*] [ASSUMING *constraint*] [*k* TIMES]

The WHERE clause is interpreted as a constraint to test against all members of the population that have been observed so far. If it is not supplied, the SIMULATE command is executed against an arbitrary as-yet-unobserved member of the population, i.e. a unique row id from the standpoint of the GPM interface. The ASSUME clause is interpreted as an additional set of constraints to condition each row on before generating the simulations.

For example, to generate a proxy dataset of two variables `varA` and `varB`, one can write `SIMULATE varA, varB FROM population 100 TIMES`. As another example, consider the BQL command `SIMULATE varA, varD FROM population 20 TIMES WHERE varB = True AND varC IS MISSING ASSUMING varC = 3.4`. This generates 20 simulated values from $p(\texttt{varA}, \texttt{varD}|\texttt{varC} = 3.4)$ for each member of the population where `varB` is equal to `True` and `varC` is missing. This behavior may seem non-intuitive. For example, a SIMULATE invocation with `WHERE true` returns Rk rows, where R is the number of rows in the database and k is the number of output samples specified with the query. On the other hand, `WHERE false` yields an empty result set, always. However, this semantics allows SQL aggregates to reduce the predictions for individual source rows by grouping on the row identifiers.

To formally describe the meaning of simulate, we first introduce some notation. Let $w(\{x_{(r,c)}|c \in \mathcal{G}\})$ be the predicate denoted by the WHERE clause, i.e. $w(\cdot) = 1$ if the predicate is satisfied and 0 otherwise. Let R be the set of rows for which there is at least one measurement, i.e. $R = \{r_i|(r_i, \cdot, \cdot) \in O\}$, and let $W = \{r_i|w(\{x_{(r_i,c)}|c \in \mathcal{G}\}) = 1\}$ be the set of rows that satisfy the WHERE clause's filter. If a WHERE clause is not provided, then $w(\{x_{(r,c)}|c \in \mathcal{G}\}) = 0$ for all existing rows $r \in R$, and $W = \{r^*\}$ be a set containing a single distinguished row about which no measurements are known. Let $T = \{c_i\}$ be the set of target columns, and let $A = (c_j, x_{(r,c_j)})$ be the set of assumed equality constraints. Also let $TA = T \cup \{c|(c, \cdot) \in A\}$ be the set of all columns referenced in the SIMULATE command.

For each $r^* \in W$, the SIMULATE primitive produces a set of k returned realizations $S_{r^*} = \{s_i\}$ of the following generative process:

$$\mathcal{G}^i \sim Discrete(\{\mathcal{G}_j\}; w_j\})$$
$$s_i \sim \{X_{r^*,c}|c \in T\}|\{X_{r^*,c'} = x_{c'}|(c', x_{c'}) \in A\}, \Theta^i, \{z^i_{(r^*,c_m)}|c_m \in TA\}$$

This corresponds to choosing a GPM at random according to the probabilities given by their weights and then generating s_i from the conditioned distribution in that model. If the models are equally weighted, i.e. $w_i = 1$, and if all the GPMs are drawn from their posterior distribution given the observations $p(\mathcal{G}|O)$, then this procedure implements sampling from the Bayesian posterior predictive distribution over the targets given all the observed data plus the additional constraints from the ASSUME clause:

$$\begin{aligned} &p(\{X_{r^*,c}|c \in T\}|\{X_{r^*,c'} = x_{c'}|(c', x_{c'}) \in A\}, O) \\ &\propto p(\{X_{r^*,c}|c \in T\}|\{X_{r^*,c'} = x_{c'}|(c', x_{c'}) \in A\}|\mathcal{G})p(\mathcal{G}|O) \end{aligned}$$

3.2.2 `ESTIMATE`: approximating posterior averages.

The second core BQL primitive, called `ESTIMATE`, allows clients to query the posterior expectations of stochastic functions that are defined over the rows and the columns:

```
ESTIMATE target properties FROM [COLUMNS OF] table [WHERE row/col filter]
```

Row-wise estimands provided by BQL. Consider the case where the rows are being queried, i.e. `COLUMNS OF` does not occur in the query. Let $P = \{f_i(x_{(r,c_i)}, \mathcal{G})\}$ be the set of properties whose values are requested. These properties can depend on observed measurements as well as latent components of the GPM. Let $w(\cdot)$ implement the `WHERE` clause's filter, as with `SIMULATE`. If a `WHERE` clause is not provided, then $w(\{x_{(r,c)} | c \in \mathcal{G}\}) = 0$ for all existing rows $r \in R$.

Given these definitions, each row in the output of this class of `ESTIMATE` invocations is defined as follows:

$$\{e_i\} \text{ with } e_i = \sum_k w_k f_i(x_{(r,c_i)}, \mathcal{G}_k)$$

The total set of returned rows is defined by the where clause:

$$\{\{e_i\}_r\} \text{ for } r \in W = \{r_i | w(\{x_{(r_i,c)} | c \in \mathcal{G}\}) = 1\}$$

1. $\log p = \text{predictive-probability}(\mathcal{G}, \texttt{row} = r, \texttt{col} = c)$

   This estimand is denoted `PREDICTIVE PROBABILITY OF` *`col`*, and applied against an implicitly specified row, thus picking out a single measurement in the population. It can be implemented by delegation to the underlying GPM:

   $$\text{predictive-probability}(\mathcal{G}, r, c)) = \text{logpdf}(\mathcal{G}, \emptyset, \{(r, c, x_{(r_j,c_j)} \text{ from } O_{\mathcal{G}})\})$$

   This can be used to identify outliers — measurements that are unlikely under their marginal distribution — as well as anomalous measurements that are marginally likely but unlikely given the other measurements for the same row.

2. $\text{sim(a,b)} = \text{generative-similarity}(\mathcal{G}, \text{context} = \{c_i\}, \text{rowA} = r^a, \text{rowB} = r^b)$

   Data analysts frequently want to retrieve rows from a table that are "statistically similar" to some pre-existing or hypothetical row. This is a key problem in data exploration. It is also useful when trying to explain surprising inference results or when trying to diagnose and repair data or inference quality issues. Many machine learning techniques treat similarity as a central primitive, and use a metric formulation of similarity as the basis for inductive generalization.

   Information theory provides appealing alternatives: measure similarity in terms of the amount of information one row contains about the values in another. This can be assessed against all variables or just against a "context" that is defined by a particular subset of variables. One approach, leading to a directional measure, is to measure the divergence of the distribution over values in one row from the distribution over values in another:

   $$Pr[\text{D}^{\mathcal{G}}_{KL}(\vec{x}_{\mathcal{G},r^a}|_{\{c_i\}} || \vec{x}_{\mathcal{G},r^b}|_{\{c_i\}}) = 0]$$

Column-wise estimands provided by BQL. Another use of `ESTIMATE` is to query properties of the columns, via `ESTIMATE ... FROM COLUMNS OF ...`. Let r^* be a distinguished row about which no measurements are known, i.e. $(r^*, \cdot, \cdot) \notin O$. Let $g_i(x_{(r^*,c)}, \mathcal{G})$ be a function of a set of measurements from a fresh row and the underlying GPM. It is then straightforward to define the set G of values needed to check the `WHERE` filter, the set of columns C_s that satisfy the filter, and the set E of returned values containing all the target expressions for each satisfying column.

$$\begin{aligned} G &= \{g_c(x_{(r^*,c)}, \{x_{(r,c)} | (r,c,\cdot) \in O\}, \mathcal{G}_k) | c \in \mathcal{G}_k\} \\ C_s &= \{g | g \in G \text{ and } w(g) = 1\} \\ E &= \cup_t \ \{g_t(x_{(r^*,c)}, \{(x_{(r,c)} | (r,c,\cdot) \in O\}, \}_k) | c \in C_s\} \end{aligned}$$

1. $p =$ marginal-dependence-prob$(\mathcal{G}, \texttt{colA} = c_i, \texttt{colB} = c_j)$

   This estimand characterizes the amount of evidence for the existence of a predictive relationship between the pair of variables c_i and c_j. It is defined according to the information-theoretic definition of conditional independence:

   $$Pr[X_{(r^*,c_i)} \perp\!\!\!\perp X_{(r^*,c_j)}] = \sum_{\mathcal{G}} Pr[I(X_{(r^*,c_i)}; X_{(r^*,c_j)}) = 0 | \mathcal{G}] Pr[\mathcal{G}]$$

   If each weighted GPM $\mathcal{G}_k$ is sampled approximately from some Bayesian posterior $Pr[\mathcal{G}|O]$ (and $w_k = 1$ identically), then simple Monte Carlo estimation of the marginal dependence probability yields an estimate of the *posterior* marginal dependence probability:

   $$Pr[X_{(r^*,c_i)} \perp\!\!\!\perp X_{(r^*,c_j)} | O]$$

2. $b =$ mutual-information$(\mathcal{G}, \texttt{colA} = c_i, \texttt{colB} = c_j)$

   The mutual information between two columns can be estimated by the standard reduction to KL divergence (Cover and Thomas, 2012). This complements the marginal dependence probability, providing one measure of the strength of a dependence.

For convenience, some of the quantities that are ordinarily accessed via `ESTIMATE` are also made available via `SELECT`.

3.2.3 `INFER`: summarizing distributions with point estimates.

Predictive modeling applications sometimes require access to point predictions rather than samples from predictive distributions. BQL provides these capabilities using the `INFER` primitive. The difference between `INFER` and `SELECT` is that `INFER` incorporates automatic implicit imputation from the underlying collection of GPMs, plus filtering based on user-specified confidence thresholds. For simplicity, this paper describes a simplified version with a single threshold:

```
INFER target columns FROM table [WHERE row filter] WITH CONFIDENCE confidence
level
```

This operation returns a set of measurements $\{x^{inf}_{(r,c)}\}$ where unobserved measurements are filled in with point estimates $\hat{x}_{(r,c)}$ if a prescribed confidence threshold $p(\text{conf}(X_{(r,c)} = \hat{x}_{(r,c)}) \geq q)$ is reached. More formally:

$$x^{inf}_{(r,c)} = \begin{cases} x_{(r,c)} & \text{foreach } (r,c,\cdot) \in O \\ \hat{x}_{(r,c)} & \text{foreach } (r,c,\cdot) \notin O \text{ and } p(\text{conf}(X_{(r,c)} = \hat{x}_{(r,c)}) \geq q) \\ \text{null} & \text{otherwise} \end{cases}$$

For discrete measurements, BQL implements conf$(\cdot)$ in terms of predictive probability:

$$\text{conf}(X_{(r,c)} = \hat{x}_{(r,c)}) = p(X_{(r,c)} = \hat{x}_{(r,c)}) = \sum_{\mathcal{G}} p(X_{(r,c)} = \hat{x}_{(r,c)} | \mathcal{G}) p(\mathcal{G}) = \sum_{\mathcal{G}} p(X_{(r,c)} = \hat{x}_{(r,c)} | \mathcal{G}) w_i$$

Optimal candidate estimates can be found by optimization, implemented via enumeration:

$$\hat{x}_{(r,c)} = \arg\max_x p(X_{(r,c)} = x)$$

For continuous measurements, there is no canonical definition of confidence that applies to all GPMs. Here we define $conf(X_{(r,c)} = x) = q$ as the probability that there is a useful unimodal summary of the distribution of $X_{(r,c)}$ that captures at least $100q$ percent of the predictive probability mass. This can be formalized in terms of mixture modeling. Let ϕ_l be the parameters of mixture component l; for continuous data, we will use Gaussian component models, so $\phi_l = (\mu_l, \sigma_l)$. Let π_l be the mass associated with component l. We will choose $conf(\cdot)$ and $\hat{x}$ as follows:

$$\begin{aligned} \{(\phi_l, \pi_l)\} &\sim p(\{(\phi_l, \pi_l)\} | \{X^k_{(r,c)}\}) \text{ for } 0 \leq k \leq K^+ \\ l^* &= \arg\max_l \pi_l \\ \hat{X}_{(r,c)} &= \mu_{l^*} \\ conf(X_{(r,c)} = \hat{x}_{(r,c)}) &= \pi_{l^*} \end{aligned}$$

Note that this approach can recover the behavior of the chosen strategy for discrete data by using component models that place all their probability mass on single values. The current prototype implementation of BayesDB uses a standard Gibbs sampler for a Dirichlet process mixture model (Mansinghka et al., 2015; Neal, 1998; Rasmussen, 2000) to sample $\{(\phi_l, \pi_l)\} | \{X^k_{(r,c)}\}$, with $K^+ = 1000$ by default. Adjusting K^+ and the amount of inference done in this mixture model can yield a broad class of tradeoffs between time, accuracy, and variance; the current values are chosen for simplicity.

3.3 Model and data independence

Relational databases revolutionized the processing and analysis of business data by enabling a single centrally managed data base to shared by multiple applications and also shared between operational and analytic workloads. This in turn accelerated the development of high performance and efficient databases, because the common abstraction became a target for researchers and industrial practitioners looking to build high performance system software with a broad impact. The relational model enabled sharing and infrastructure reuse because interactions with the data, queries, are expressed in a notation (most popularly SQL) that is independent of the physical representation of the data (Codd, 1970). Without this independence, physical data layout must be carefully tailored to particular workloads, specialized code written to manipulate the layout, and these data formats and access methods cannot easily be shared.

BayesDB aims to provide additional abstraction barriers that insulate clients from the statistical underpinnings of data analysis. Clients need to be able to specify data analysis steps and workflows in a notation that is independent of the models and runtime inference strategies used to implement individual primitives, and (where possible) the modeling strategies used to produce models from the original data.

Recall that the complete persistent state of a single population in BayesDB is characterized by two mathematical objects:

1. The complete set of observed measurements $O = \{(r_i, c_i, x_{(r_i,c_i)})\}$.
2. The weighted collection of GPMs $\{(w_k, \mathcal{G}_k)\}$. Note that this notation makes no commitment as to the content of the GPMs, the weights, or the procedures by which they were obtained.

The independencies provided by BayesDB can be described in terms of these objects:

1. *Physical data independence.* The notation for O makes no commitment as to the physical representation of the measurements. The definitions of BQL primitives given above therefore do not depend on details of the data representation to define their values. However, as with SQL, small changes in representation may yield large changes in runtime performance.
2. *Physical model independence.* The notation for each $(w_k, \mathcal{G}_k)$ makes no commitment as to the specific probability distribution induced over the set of random measurements $X = \{X_{(r_i,c_i)}\}$. The definitions of BQL primitives given above therefore do not depend on the details of the probabilistic models used to define the random result set for each query. In principle, the mathematical properties of the models as well as their software implementation (or even implementing platform) can be changed without invalidating end user queries. However, small changes in the generative population model may yield large changes in the results of *simulate* and *logpdf*.

Databases provide other finer-grained independence properties that may have useful analogs in BayesDB. For example, let us partition the random variables induced by a given GPM into two subsets $X^A = \{X_{(r_i^a, c_i^a)}\}$ and $X^B = \{X_{(r_j^b, c_j^b)}\}$. An example of a desirable data-dependent independence property is that if $X^A|O \perp\!\!\!\perp X^B|O$ in the "true" GPM, then $Q|X^A, X^B = Q|X^A$ in any inferred models. Informally, this rests on the model-building strategy: if the model-builder recovers the correct independencies, then the independence of query results follows. This can be thought of as an analogue of logical data independence, which stipulates that e.g. adding new features should not affect the behavior of existing applications whose results do not depend on the value of these new features. Formalizing and verifying these properties is an important challenge for future research.

4. Modeling with the Meta-Modeling Language

BayesDB also provides the Meta-Modeling Language (MML), a probabilistic programming language for building models of data tables. MML programs consist of *modeling tactics* that control the behavior of an automatic model-building engine. These tactics take several forms: statistical datatypes; initialization of weighted collections of random models; approximately Bayesian updating of the model collection; qualitative assertions about dependence and independence between variables; and the use of custom statistical models for specific conditional distributions. All these tactics are currently implemented in terms of a unifying semi-parametric Bayesian model that fills in all unspecified aspects.

4.1 Statistical datatypes

This metadata constrains the probability models that will be used for each column of data and can also be used to choose appropriate visualizations. It is straightforward to support several different kinds of data:

1. *Categorical values from a closed set.* This datatype includes a dictionary that maps the raw data values (often strings) to canonical numerical indexes for efficient storage and processing. This information can also be used to inform modeling tactics. For example, in the current version of MML, closed-set categorical variables are modeled generatively via a multinomial component model with a symmetric Dirichlet prior on the parameters (Mansinghka et al., 2015). Discriminative models for closed-set categorical columns could potentially use a multinomial logit link function, or an appropriate multi-class classification scheme.

2. *Binary data.* Data of this type is generatively modeled using an asymmetric Beta-Bernoulli model (Mansinghka et al., 2015) that can better handle sparse or marginally biased variables than a symmetric alternative. Also, a broad class of discriminative learning techniques can natively handle the binary classification problems induced by binary variables.

3. *Count data.* Non-negative counts can be naturally modeled generatively by a Poisson-Gamma model or discriminatively by a GLM with the appropriate link function.

4. *Numerical data.* By default, data of this type is generatively modeled using a standard Normal-Gamma model. It is straightforward to add numerical ranges to enforce truncation post-hoc, and to add numerical pre-transformations that are appropriate for data that is naturally viewed as normal only on a log scale.

We have performed preliminary experiments on other datatypes built on standard statistical models. For example, cyclic data can be handled via a von Mises model (Gopal and Yang, 2014). Many other datatypes can be handled by the appropriate generalized linear model (McCullagh and Nelder, 1989). Broadening the set of primitive data types and assessing coverage on a representative corpus of real-world databases will be a key research challenge going forwards.

4.2 Bayesian generative population meta-models

Some data generators can be learned from data. Often the learning mechanism will be based on approximate probabilistic inference in a *meta-model*: a probabilistic model defined over a space of data generators, each of which is also a probabilistic model in its own right. Thus far, all BayesDB meta-models have been *Markov chain meta-models*. These meta-models internally maintain a single sample from an approximate posterior, and provide a Markov chain transition operator that updates this sample stochastically.

1. $\mathcal{G} = (\theta^0_{\mathcal{G}}, \mathbf{X}_{\mathcal{G}}) = \text{initialize}(\text{meta-schema} = \Lambda)$

   Initializes a new meta-model with arbitrary parameters and an associated tabular data store.

2. $\text{incorporate}(\text{id} = r, \text{ values} = \{(c_j, x_{(r,c_j)})\})$

   Creates a new member of the population with the given row index and values and stores it. Errors result from duplicate indexes or variables c_j whose values $x_{(r,c_j)}$ are not compatible with the meta-schema Λ (e.g. because the expected data type is incompatible with a provided value).

3. $\text{remove}(\text{id} = r)$

   Removes a member of the population from the data store.

4. $\text{infer}(\text{program} = \mathcal{P})$

   Simulate an internal Markov chain transition operator $\mathcal{T}$ to improve the quality of the current sampled model representation:

$$\theta^{i+1}_{\mathcal{G}} = \mathcal{T}(\theta^i_{\mathcal{G}})$$

Some Markov chain meta-models are *asymptotically Bayesian*, i.e. the distribution that results from sequences of T updates converges to the posterior over meta-models as T goes to infinity:

$$\lim_{t\to\infty} \mathrm{D}_{KL}(p(\theta_{\mathcal{G}}|\mathbf{X}_{\mathcal{G}})||p(\mathcal{T}^t(\theta_{\mathcal{G}}))) = 0$$

A sufficiently expressive Markov chain meta-model may also be asymptotically consistent in the usual sense. The default semi-parametric GPM provided by BayesDB is designed to be both asymptotically consistent and asymptotically Bayesian; these invariants are crucial for its robustness, broad applicability, and suitability for use by non-experts. Formally specifying and validating these properties is an important challenge for future research.

4.2.1 Controlling inference via INITIALIZE and ANALYZE.

The MML allows users to control the process by which models are created and updated to reflect the data. These capabilities are exposed via two commands:

1. `INITIALIZE` *k* `MODELS FOR` *population*

   This command creates models by sampling their structure and parameters from the underlying GPM's prior. This is implemented by delegation to `initialize(`Λ`)` where Λ is the entire MML schema so far.

2. `ANALYZE [`*variable subset* `OF]` *population* `FOR` *timelimit*

   This command performs approximately Bayesian updating of the models in the weighted collection by delegating to the `infer()` procedure from the underlying GPM. Here is a typical invocation:

   ```
   ANALYZE my_population FOR 10 MINUTES
   ```

   When no variable subset is provided, analysis is done on all the latent variables associated with every GPM in the weighted collection. Finer-grained control is also possible using variable subset specifiers that pick out particular portions of the latent state in the GPM; these details are beyond the scope of this paper.

4.3 Qualitative constraints

The BayesDB MML provides constructs for specifying qualitative constraints on the dependence and independence relationships (Pearl, 1988). The model-building engine attempts to enforce them in all GPMs[1]. These constraints are specified as follows:

```
ALTER METAMODEL FOR population ENSURE colA IS [NOT] MARGINALLY DEPENDENT
ON colB
```

It is also possible to `INITIALIZE` and `ANALYZE` models that do not respect the constraints, and then enforce them after the fact:

```
ALTER MODELS FOR population ENSURE colA IS [NOT] MARGINALLY DEPENDENT
ON colB
```

These commands enable domain experts to apply qualitative knowledge to make better use of sparse data. This can be crucial for improving analysis and model credibility in the eyes of domain experts. They also create the opportunity for false or unjustified knowledge to influence the results of analysis. This can reduce credibility in the eyes of statisticians or domain skeptics who want to see all assumptions in an analysis scrutinized empirically.

1. The current implementation does not attempt to detect contradictions.

4.4 Incorporating foreign statistical models

A crucial aspect of MML is that it permits experts to override the automatic model-building machinery using custom-built statistical models. Feedforward networks of such models can be specified as follows:

```
ALTER SCHEMA FOR population MODEL output variables GIVEN input variables
USING FOREIGN PREDICTOR FROM source file
```

Presently these models are presumed to be discriminative. They are only required to be able to simulate from a probability distribution over the output variables conditioned on the inputs, and to evaluate the probability density induced by this distribution.

4.5 A semi-parametric factorial mixture GPM

The current implementation of MML implements all the above commands in terms of approximate inference in single, unusually flexible, semi-parametric Bayesian meta-model. This GPM is closely related to CrossCat (Mansinghka et al., 2015). The CrossCat model is a factorial Dirichlet process mixture model, where variables are assigned to specific Dirichlet process mixtures by inference in another Dirichlet process mixture model over the columns. The version used for implementing MML adds two key components:

1. *Deterministic constraints on model structure.* Users can specify constraints on the marginal dependence or independence of arbitrary pairs of variables.

2. *Feedforward networks of discriminative models conditioned on the outputs of the generative model.* This allows users to combine general-purpose density estimation with standard statistical techniques such as regression as well as complex computational models with noisy outputs.

Thus in MML, the CrossCat probability model is used as the root node in a directed graphical model. Each other node in the graph corresponds to specific discriminative model, directly conditioned on the inputs of its immediate ancestors. Undirected terms attached to the root node enforce deterministic constraints.

It is helpful to view this model in terms of a “divide and conquer” modeling strategy that bottoms out in foreign predictors and other standard parametric models from Bayesian statistics:

1. All variables not explicitly assigned to a custom model are divided into marginally independent groups. Variables in the same group are assumed to be marginally dependent. Partitions of variables that do not respect the given marginal dependence and independence constraints are rejected. Each group of variables induces an independent subproblem that will typically be far lower dimensional than the original high-dimensional problem.

2. For each subproblem, divide the rows into clusters whose values are marginally dependent given any variable-specific hyperparameters.

3. For each cluster, use a simple product of parametric models — i.e. a "naive Bayes" approach (Duda et al., 2001) — to estimate the joint distribution.

Inference in the GPM thus addresses modeling tradeoffs that resemble the decisions faced in exploratory analysis, confirmatory analysis, and predictive modeling. The most crucial decisions involve defining which subset of the data is relevant for answering each question. A secondary issue is what probabilistic model to use for each subset; absent prior knowledge, these are chosen generically, based on the type of the data in the column.

4.5.1 A "divide-and-conquer" generative process

The generative process that induces the default GPM can be described using the following notation:

Name	**Description**
α_D	Concentration hyperparameter for CRP that slices the columns
$\vec{\lambda}_d$	Hyperparameters for column d (datatype-dependent)
z_d	Slice (column partition) assigned to column d
α_v	Concentration hyperparameter for CRP that clusters rows for slice v
y_r^v	Cluster assigned to row r with respect to slice v
$\vec{\theta}_c^d$	Model parameters for column d cluster c (datatype-dependent)
$\vec{x}_{(\cdot,d)}^c$	Values in cluster c for column d, i.e. $\{x_{(r,d)} \mid y\|_r^{z_d} = c\}$
u_d	An indicator such that $u_d = 1$ iff d is modeled by a foreign predictor
$par(d)$	The set of input dimensions for the foreign predictor conditionally modeling variable d
$\vec{\phi}_d$	Parameters for the foreign predictor conditionally modeling variable l
$m_d(x_{(r,d)}; \vec{\phi}_d, \vec{x}_p)$	The stochastic model for the foreign predictor used for variable d (with density $m_d^{dens}(\cdot)$) with $\vec{x}_p = \{x_{(r,p)} \| p \in par(d)\}$)
$\delta_m \vec{z}$	Characteristic function enforcing marginal (in)dependence constraint m
$V_d(\cdot)$	A generic hyper-prior of the appropriate type for variable or dimension d.
$M_d(\cdot)$ and $L_D(\cdot)$ $\forall\ d\ s.t.\ u_d = 1$	A datatype-appropriate parameter prior (e.g. a Beta prior for binary data, Normal-Gamma for continuous data, or Dirichlet for discrete data), and likelihood model (e.g. Bernoulli, Normal or Multinomial).

Using this notation, the unconstrained generative process for the default meta-model can be concisely described in statistician's notation as follows:

$$
\begin{aligned}
\alpha_D &\sim \text{Gamma}(k=1, \theta=1) && \\
\vec{\lambda}_d &\sim V_d(\cdot) && \text{foreach } d \in \{1, \cdots, D\} \\
z_d &\sim \text{CRP}(\{z_i \mid i \neq d\}; \alpha_D) && \text{foreach } d \in \{1, \cdots, D\} \\
\alpha_v &\sim \text{Gamma}(k=1, \theta=1) && \text{foreach } v \in \vec{z} \\
y_r^v &\sim \text{CRP}(\{y_i^v \mid i \neq r\}; \alpha_v) && \text{foreach } v \in \vec{z} \text{ and} \\
& && r \in \{1, \cdots, R\} \\
\vec{\theta}_c^d &\sim M_d(\cdot; \vec{\lambda}_d) && \\
\vec{x}_{(\cdot,d)}^c = \{x_{(r,d)} \mid y_r^{z_d} = c\} &\sim \prod_r L_d(\vec{\theta}_c^d) && \text{if } u_d = 0 \\
\vec{x}_{(\cdot,d)} = \{x_{(r,d)}\} &\sim m_d(\vec{\phi}_d; \{x_{(r,p)} | p \in par(d)\}) && \text{if } u_d = 1 \\
c_m &\sim \delta_m(\vec{z}) && \text{foreach (in)dependence constraint}
\end{aligned}
$$

The true generative process also must ensure that $c_m = 1$ for all of the M (in)dependence constraints. This is enforced by conditioning on the event $\{c_m = 1\}$. A generative model for this constrained process can be given trivially by embedding the unconstrained generative process in the inner loop of a rejection sampler for $\{c_m\}$ (Mansinghka, 2009; Murray et al., 2009).

4.5.2 The joint density

Here we use $\theta_{\mathcal{G}}$ to denote all the latent information in a semi-parametric GPM $\mathcal{G}$ needed to capture its dependence on the data O. This includes the concentration parameter α_D for the CRP over columns, the variable-specific hyper-parameters $\{\vec{\lambda}_d\}$, the column partition $\vec{z}$, the column-partition-specific concentration parameters $\{\alpha_v\}$ and row partition $\{\vec{y}^v\}$, and the category-specific parameters $\{\vec{\theta}_c^d\}$. Note that in this section, M_d, V_d, L_d, and CRP each represent probability density functions rather than stochastic simulators.

Given this notation, we have:

$$
\begin{aligned}
P(\theta_{\mathcal{G}}, O) &= P(\mathbf{X}, \{\vec{\theta}_c^d\}, \{\vec{y}^v, \alpha_v\}, \{\vec{\lambda}_d\}, \vec{z}, \alpha_D) \\
&= e^{-\alpha_D} \big(\prod_{d \in D} V_d(\vec{\lambda}_d) \big) \text{CRP}(\vec{z}; \alpha_D) \big(\prod_{v \in \vec{z}} e^{-\alpha_v} \text{CRP}(\vec{y}^v; \alpha_v) \big) \\
&\times \big(\prod_{v \in \vec{z}} \prod_{c \in \vec{y}^v} \prod_{d \in \{i \text{ s.t. } z_i = v\}} M_d(\vec{\theta}_c^d; \vec{\lambda}_d) \prod_{r \in c} L_d(x_{(r,d)}; \vec{\theta}_c^d) \big) \big(\prod_m \delta_m \vec{z} \big) \\
&\times \big(\prod_{d \text{ with } u_d = 1} \prod_r m_d^{dens}(x_{(r,d)}; \vec{\phi}_d, \{x_{(r,p)} | p \in par(d)\}) \big)
\end{aligned}
$$

4.5.3 Inference via sequential Monte Carlo with Gibbs proposals and Gibbs rejuvenation

Inference in this meta-model is performed via a sequential Monte Carlo scheme, in which each row is incorporated incrementally, with all latent variables proposed from their conditional distribution. Additionally, clients can control the frequency and target latent variables for rejuvenation kernels based on Gibbs sampling, turning the overall scheme into a resample-move algorithm (Andrieu et al., 2003; Smith et al., 2013). This combination enables parallel inference and estimation of marginal probabilities while allowing the bulk of the inferential work to be done via a suitable Markov chain.

1. incorporate(id $= r$, values $= \{(c_j, x_{(r,c_j)})\}$)

   Each row is incorporated via a single Gibbs step that numerically marginalizes out all the latent variables associated with the row (Smith et al., 2013; Murphy, 2002). The associated weight is the marginal probability of the measurements to be incorporated:

   $$w_i' = w_i * p(\{(c_j, x_{(r,c_j)})\}|\mathcal{G}_\rangle)$$

   This operation is linear in the number of observed cells for the record being incorporated, the number of total slices, and the maximum number of clusters associated with any slice.

2. infer(iterations $= N$, type $=$ rows | columns | parameters | hyperparameters | foreign | resample, slice $= j$ | NA, cluster $= k$ | NA, foreign_predictor $= l$ | NA)

   This operation applies a particular transition operator, specified by the arguments, to a selected subset of the latent variables. Each invocation affects all particles in the sequential Monte Carlo scheme. By varying the `type` parameter, a client can control whether inference is performed over the row-cluster assignment variables, the column-slice assignment variables, the cluster parameters, the column-specific hyperparameters, or all latent variables associated with a specific foreign predictor. An invocation with `type = resample` applies multinomial resampling to the weighted collection of models.

   This allows for a limited form of inference programming (Mansinghka et al., 2014), as follows. By varying the `slice`, `cluster`, or `foreign_predictor` variables, clients can instruct the GPM to only perform inference on a specific subset of the latent variables. Computational effort can thus be focused on those latent variables that are most relevant for a given analysis, rather than uniformly distributed across all latent variables in the GPM. This is most useful when the queries of interest focus on a subset of the variables, or when the clusters are well-separated.

   The prototype implementation of BayesDB uses row-cluster, column-slice, cluster-parameter, and column-hyperparameter transition operators from Mansinghka et al. (2015). The only modification is that the log joint density now includes terms for enforcing each of the (in)dependence constraints, and also terms for the likelihood induced by each foreign predictor, as described above.

This interface allows clients to specify multiple MCMC, SMC and hybrid strategies for inference. The default inference program that is invoked by the `ANALYZE` command in BQL does no resampling and selects slices and clusters to do inference on via systematic scans. It thus can be thought of as an MCMC scheme with multiple parallel chains. This approach is conservative and makes it easier to assess the stability and reproducibility of inference, although it is unlikely to be the most efficient approach in some cases.

5. Discussion

This paper has described BayesDB, a probabilistic programming platform that allows users to directly query the probable implications of statistical data. The query language can solve statistical inference problems such as detecting predictive relationships between variables, inferring missing values, simulating probable observations, and identifying statistically similar database entries. Statisticians and domain experts can incorporate (in)dependence constraints and custom models using a qualitative language for probabilistic models. The default meta-model frees users from needing to know how to choose modeling approaches, remove records with missing values, detect outliers, or tune model parameters. The prototype implementation is suitable for analyzing complex, heterogeneous data tables with up to tens of thousands of rows and hundreds of variables.

5.1 Related work in probabilistic programming

Most probabilistic programming languages are intended for model specification (Goodman et al., 2008; Stan Development Team, 2015; Milch et al., 2007; Pfeffer, 2009). This is fundamentally different from BQL and MML:

1. In BQL, probabilistic models are never explicitly specified. Instead, an implicit set of models is averaged over (or sampled from) as needed.

2. With MML, users specify constraints on an algorithm for model discovery and need not explicitly select any specific models. These constraints generally do not uniquely identify the structure of the model that will ultimately be used.

In contrast, with languages such as Stan (Stan Development Team, 2015), each program corresponds to a specific probabilistic model whose structure is fixed by the program source. Tabular (Gordon et al., 2014), a probabilistic language designed for embedding into spreadsheets that applies user-specified factor graph models defined in terms of observed and latent variables to datasets represented as sub-tables, seems closest in structure to BQL. However, like BUGS and Stan, Tabular does not aim to hide the conceptual vocabulary of probabilistic modeling from its end users, and it focuses on user-specified models. Other integrations of probabilistic modeling with databases such as (Singh and Graepel, 2013) are similarly focused on sophisticated modeling but do not provide a model-independent abstraction for queries or support for general Bayesian data analysis.

It is straightforward to extend MML to allow syntactic escapes into all these languages that allow external probabilistic programs to be used as foreign predictors.

5.2 Related work in probabilistic databases

BayesDB takes a complementary approach to several recent projects that integrate aspects of probabilistic inference with databases. The most closely related systems are MauveDB (Deshpande and Madden, 2006) and BBQ (Deshpande et al., 2004). They provide *model-based views* that enable users to run standard SQL queries on the outputs of statistical models. These models must be explicitly specified as part of the schema. This is useful for some machine learning applications but does not address the core problems of applied inference, such as data exploration, data cleaning, and confirmatory analysis. Both systems also use restricted model classes that can easily introduce substantial for ad-hoc predictive queries.

Other systems such as MLBase (Kraska et al., 2013) and GraphLab (Low et al., 2012) aim to simplify at-scale development and deployment of machine learning algorithms. ML-Base and GraphLab host data in a distributed database environment and provide operators for scalable ML algorithms. Systems such as SimSQL (Cai et al., 2013) and its ancestor, MCDB (Jampani et al., 2008), provide SQL operators for efficient Monte Carlo sampling. In principle, several of these systems could serve as runtime platforms for optimized implementations of BQL and the MML.

5.2.1 Uncertain data versus uncertain inference

The database research community has proposed several probabilistic databases that aim to simplify the management and querying of data that is "uncertain" or "imprecise" (Dalvi et al., 2009). This "data uncertainty" is different from the inferential uncertainty that motivates BayesDB. Even when the data is known with certainty, it is rarely possible to uniquely identify a single model that can be used with complete certainty. Second, each probable model is likely to have uncertain implications. Extensions of BayesDB that augment GPMs with probabilistically coherent treatments of data uncertainty are an important area for future research.

5.3 Limitations and future work

Additional GPMs and meta-models are needed for some applications. There are specialized SQL databases that strike different tradeoffs between query latency, workload variability, and storage efficiency. Similarly, we expect that future GPMs and meta-models will strike different tradeoffs between prediction speed, prediction accuracy, statistical model capacity, and the amount of available data. In some cases, the semi-parametric meta-model presented here may be adequate in principle but producing an appropriate implementation is a significant systems research project. For example, it may be possible to build versions suitable for ad-hoc exploration of distributed databases such as Dremel (Melnik et al., 2010) or Spark (Zaharia et al., 2010). In other cases, fundamentally different model classes may be more appropriate. For example, it seems appealing to jointly model populations of web browsing sessions and web assets with low-dimensional latent space models (Stern et al., 2009).

It will be challenging to develop query planners that can handle GPMs given by arbitrary probabilistic programs. A key issue is that the full GPM interface allows for complex conditional queries over composite GPMs that may require data-dependent inference strategies. One potential approach is to specify GPMs as probabilistic programs in a language with programmable inference; currently, the only such language is VentureScript. The inference strategy needed to answer a given query could then be assembled on-demand.

BQL and MML have yet to incorporate key ideas from several significant subfields of statistics. For example, neither language has explicit support for causal relationships and arbitrary counterfactuals (Pearl, 1988, 2009, 2001). Both BQL and MML make the standard, simplistic assumption that data is missing at random. Neither BQL nor MML has native support for longitudinal or panel data or for time-series; instead, users must apply standard workarounds or implement custom data types. A minor limitation is that hierarchical models are currently supported by merging subpopulations, retaining an indicator variable, and treating any variables unique to a given subpopulation as missing. It should instead be possible to build GPMs that jointly model subpopulations that are separately represented (and that therefore may not share the same set of observable variables). It will also be important to develop a formal semantics and cost model for both BQL and MML.

Qualitative probabilistic programming. BQL and MML are qualitative languages for quantitative reasoning. They make it possible for users to perform Bayesian data analysis without needing to know how to specify quantitative probabilities or model parameters. However, the set of qualitative constructs that they support is limited, and needs to be expanded. For example, in MML, it will be important to support conditional dependence constraints. These could be specified generatively, e.g. by defining a directed acyclic graph over subsets of variables, and leaving the model builder to fill in the (conditional) joint distributions over each subset of variables. In BQL, it would be interesting to explore the addition of commands for optimization and decision-theoretic choice, with objective functions specified both explicitly and implicitly. Finally, it will be interesting to explore elicitation strategies based on "programming by example". For example, users could create datasets by iteratively specifying prototypical examples and turn them into large datasets by treating each as the seed for a separate synthetic population, produced via `SIMULATE`.

5.4 Conclusion

Traditional databases protect consumers of data from "having to know how the data is organized in the machine" Codd (1970) and provide automated data representations and retrieval algorithms that perform well enough for a broad class of applications. Although this abstraction barrier is only imperfectly achieved, it has proved useful enough to serve as the basis of multiple generations of software and data systems. This decoupling of task specification from implementation made it possible to improve performance and reliability — of individual database indexes, and in some cases of entire database systems — without needing to notify end users. It also created a simple conceptual vocabulary and query language for data management and data processing that spread far farther than the systems programming knowledge needed to implement it.

BayesDB aims to insulate consumers of statistical inference from the concepts of modeling and statistics and provide a simple, qualitative interface for solving problems that currently seem quantitative and complex. It also allows models, analyses, and data resources to be improved independently. It is not yet clear how deeply the analogy with traditional databases will run. However, we hope that BayesDB represents a significant step towards making statistically rigorous empirical inference more credible, transparent and ubiquitous.

References

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Zhuhua Cai, Zografoula Vagena, Luis Perez, Subramanian Arumugam, Peter J Haas, and Christopher Jermaine. Simulation of database-valued markov chains using simsql. In

Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pages 637–648. ACM, 2013.

Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

Amol Deshpande and Samuel Madden. Mauvedb: supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 73–84. ACM, 2006.

Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 588–599. VLDB Endowment, 2004.

R.O. Duda, P.E. Hart, D.G. Stork, et al. *Pattern classification*, volume 2. wiley New York, 2001.

Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall, London, 1995.

Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence*, 2008.

Siddharth Gopal and Yiming Yang. von mises-fisher clustering models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 154–162, 2014.

Andrew D Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. Tabular: a schema-driven probabilistic programming language. In *ACM SIGPLAN Notices*, volume 49, pages 321–334. ACM, 2014.

Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 687–700. ACM, 2008.

Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua Tenenbaum. Crosscat: A fully bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *Journal of Machine Learning Research*, 2015.

Vikash Kumar Mansinghka. *Natively probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, 2009.

Peter McCullagh and John A Nelder. *Generalized linear models*, volume 37. CRC press, 1989.

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. 1 blog: Probabilistic models with unknown objects. *Statistical relational learning*, page 373, 2007.

Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

Iain Murray, David MacKay, and Ryan P Adams. The gaussian process density sampler. In *Advances in Neural Information Processing Systems*, pages 9–16, 2009.

R. Neal. Markov chain sampling methods for dirichlet process mixture models, 1998.

Judea Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann, San Mateo, California, 1988.

Judea Pearl. Bayesianism and causality, or, why i am only a half-bayesian. In *Foundations of bayesianism*, pages 19–36. Springer, 2001.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137, 2009.

C. Rasmussen. The infinite gaussian mixture model. In *Advances in Neural Processing Systems 12*, 2000.

Stuart Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Prentice Hall, Upper Saddle River, New Jersey, 2003.

Sameer Singh and Thore Graepel. Automated probabilistic modeling for relational data. In *Proceedings of the ACM of Information and Knowledge Management CIKM 2013*. ACM, 2013. URL `http://research.microsoft.com/apps/pubs/default.aspx?id=200220`.

Adrian Smith, Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo methods in practice*. Springer Science & Business Media, 2013.

Stan Development Team. *Stan Modeling Language Users Guide and Reference Manual, Version 2.8.0*, 2015. URL `http://mc-stan.org/`.

David H Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, pages 111–120. ACM, 2009.

L. Wasserman. Low assumptions, high dimensions. *Rationality, Markets and Morals*, 2, 2011.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.