Regression Databases: Probabilistic Querying using Sparse Learning Sets

Alexander Brodsky
George Mason University
brodsky@gmu.edu

Carlotta Domeniconi
George Mason University
carlotta@ise.gmu.edu

David Etter
George Mason University
detter@gmu.edu

Abstract

We introduce Regression Databases (REDB) to formalize and automate probabilistic querying using sparse learning sets. The REDB data model involves observation data, learning set data, views definitions, and a regression model instance. The observation data is a collection of relational tuples over a set of attributes; the learning data set involves a subset of observation tuples, augmented with learned attributes, which are modeled as random variables; the views are expressed as linear combinations of observation and learned attributes; and the regression model involves functions that map observation tuples to probability distributions of the random variables, which are learned dynamically from the learning data set. The REDB query language extends relational algebra project-select queries with conditions on probabilities of first-order logical expressions, which in turn involve linear combinations of learned attributes and views, and arithmetic comparison operators. Such capability relies on the underlying regression model for the learned attributes. We show that REDB queries are computable by developing conceptual evaluation algorithms and by proving their correctness and termination.

1 Introduction

Decision makers are often confronted with the difficult task of reasoning based on sparse learning sets. Decisions for a drug company about which lines of R&D to pursue or move to clinical trials, are often made using limited testing data and expert opinion. Terrorist threats assessment to determine the vulnerability of a group of locations to specific types of attacks, is another area where limited samples and expert opinion are the only means available to make critical decisions.

To deal with this kind of reasoning we introduce and automate the methodology of (1) identifying a set of "entities" which may represent scenarios, events or courses of action to be considered; (2) associating each entity with a set of "observable" attributes that can be precisely determined; (3) associating each entity with additional "learned" attributes, which can be estimated only probabilistically as a function of the observable attributes; and (4) querying and reasoning about the entities based on user-posed conditions with user-specified levels of confidence.

Extensive work has been done in relational databases to handle imprecise data and to reason with uncertainty. Most of the proposed approaches focuses on the estimation of unknown attribute values [7, 6, 1]. The work in [7] considered regression models and classification analysis to estimate continuous and categorical unknown attribute values. The models are generated under the assumption that statistical relationships exist between relevant attributes and the attribute with an unknown value. This work was extended in [6], where the concept of regression models was applied to database constraints. The work in [2] generalizes the OLAP model to incorporate both uncertainty and imprecise data. In general, the methods developed for the imputation of missing values assume that, typically, the value of an attribute is defined in the database, and only occasionally it is unknown. Therefore, such techniques are not suited to support the decision making process when the information on the variables of interest is actually very sparse.

Another related area of research is the prediction of attribute values using prior information [3, 9]. Specifically, this research attempts to predict sensor values or moving object locations using a prior value and other domain information. The work in [5] extended the conventional relational schema with the objective of assigning a probability value to predicates. This task is achieved by appending a special column, called *probability stamp*, which assigns a fixed probability value to each tuple. An extended relational algebra [4] and SQL syntax and semantics were introduced to handle probabilistic conditions over tuples. In this work, tuples are associated with fixed probability values assigned at users' discretion. This is an unrealistic assumption when working with massive data where the decision making process involves quantities whose dependency on the given observations cannot be formulated in a close form by domain experts. In addition, the probabilistic query language developed in previous work is limited

to handle probability conditions on tuples and/or attributes. On the other hand, complex decision making processes, often require answering queries that involve probabilities of combinations of random variables. Finally, the statistical community has proposed regression models to learn parameters of specific families of functions from given training data [8]. Such models, though, are suited to describe specific relationships, but are not sufficient for database applications, where dynamic and diverse queries, which require on-the-fly estimation of probabilities, need to be computed.

The contributions of this paper can be summarized as follows. First, we introduce and formalize a Regression Database (REDB) data model, which involves (1) known data over *observation* attributes; (2) learning set over *learned* attributes; (3) *view* attributes, defined as linear combinations of learned and observation attributes, and (4) a *regression model*. In this paper we assume that all attributes are numeric and range over $\Re$.

Second, we introduce probabilistic reasoning over REDB data by defining a query language. It extends relational algebra project-select queries with conditions on probabilities of first-order logical expressions, which involve linear combinations of learned attributes and views, and arithmetic comparison operators.

Third, we show that REDB queries are computable by developing conceptual evaluation algorithms, and by proving their termination and correctness. A key technical difficulty is the computation of probabilities of first-order logical formulae with learned attributes and views. We note that even if such a formula is in a disjunctive normal form, probabilistic events described by its disjuncts are not disjoint; and events corresponding to conjuncts in each disjunct are not independent.

2 Motivating Example

Consider the scenario of a security analyst who would like to reason about potential terrorist threat locations using conditions on attributes representing assessment measures. Table 1 is an instance of *observation* table which contains five *observation* attributes. The *Loc* attribute represents a location of a potential threat. Loss of life (Life) represents the number of people that would directly be harmed by an attack on this target. Protection level (PLev) defines a numeric weighting of the security at that location. Here, a government building or military installation would have a very high weight, where as a bridge would most likely have a very low weight. Monetary damage (MD) puts a dollar figure on the amount of damage and economic impact that would be caused by an attack on a location. Political visibility (PVis) identifies an additional weight that is used to evaluate a locations significance to public awareness. Bridges and power plants would have high weights, but we may also see fairly high weights on historic sites that could drastically effect public confidence.

Table 2 is an instance of a learning table. It consists of a sparse sample set which is a subset of the threat location table. This sample extends the threat table schema with the Likelihood of attack (LAtt) attribute, which represents an estimate, based on a survey by experts from related fields.

Table 3 shows the definition of the learned attribute (LAtt). It also defines two *view* attributes: Expected loss of life (ExLife) and Expected monetary damage (ExMD). Using these learned attributes and views we can pose queries such as "*Select the records for which the probability of the expected monetary damage is greater than 10 and less than 50 with a confidence of .95*".

Observation Attributes				
Loc	Life	MD	PLev	PVis
A07	20	62	23	10
B21	2	24	35	14
F51	30	17	11	77
G01	150	89	62	22

Table 1. Observation Table

Learning Attributes					
Loc	Life	MD	PLev	PVis	LAtt
A07	20	62	23	10	.22
B21	2	24	35	14	.67

Table 2. Learning Table

Learned Attributes	View Attributes	
LAtt= $f(Loc, Life, ..)$	ExLife= Life*LAtt	ExMD= MD*LAtt

Table 3. Learned Attributes and Views

3 Data Model

In this section we present our uncertainty assumptions in the form of regression analysis and define the schema and instances for a regression database.

3.1 Regression Analysis Preliminaries

We would like our reasoning model to allow queries, over the independent observation table, that include selection conditions which use the additional measures defined in our learning table. Here we will turn to regression analysis, which attempts to build a model based on the relationship of several independent variables and a dependent variable. We consider the technique of linear least squares regression which has been proven effective across many disciplines.

Let $O_1, \ldots, O_n$ be the names of the independent attributes, and $L_1, \ldots, L_k$ the names of the dependent attributes. The latter are random variables defined over the underlying distribution of sample tuples in $dom(O_1) \times dom(O_2) \times \cdots \times dom(O_n)$. Suppose the learning table contains m tuples. Let us denote such a tuple as $\mathbf{o}_i = (o_{i1}, \ldots, o_{in})$ for $i = 1, \ldots, m$. The collections of data $C_j = \{(\mathbf{o}_i, l_{ji})\}_{i=1}^{m}$, for $j = 1, \ldots, k$, represent the available training data to estimate the values of the random variables L_j, for $j = 1, \ldots, k$ respectively. For each training set C_j, we learn a regression model: $L_j = f_j(\mathbf{o}_i, \beta) + N_j$ for $j = 1, \ldots, k$ and $i = 1, \ldots, m$. Here N_j is a random noise distributed as a Gaussian with 0 mean and variance σ_j so that: $E[L_j] = E[f_j(\mathbf{o}_i, \beta) + N_j] = E[f_j(\mathbf{o}_i, \beta)] = f_j(\mathbf{o}_i, \beta)$, for all $j = 1, \ldots, k$ (where E is the expected value). We use the standard least squares method to find coefficients of each f_j that minimize σ_j.

3.2 Regression Database Schema and Instance

This section presents formal definitions for our regression database schema and instance.

Definition 1 (*Regression Database Schema*). A regression database schema is a quadruple $< OS, LS, VS, RS >$, where OS is an *independent observation table schema*, LS is a *learning table schema*, VS is a *view schema*, and RS is a *regression schema*.

- An observation schema OS is defined as

  $$< O_1, \cdots, O_n >$$

  where each attribute O_i, for $i = 1, \cdots, n$, has an associated domain $dom(O_i)$.

- A learning schema LS is defined as

  $$< O_1, \cdots, O_n, L_1, \cdots, L_k >$$

  where each attribute O_i, for $i = 1, \cdots, n$ is from the observation table schema, and each attribute L_j, for $j = 1, \cdots, k$, has an associated domain $dom(L_j)$.

- A view schema VS is a tuple of the form

  $$< (V_1, v_1(o_1, \cdots, o_n, l_1, \cdots, l_k)), \cdots, (V_m, v_m(o_1, \cdots, o_n, l_1, \cdots, l_k)) >$$

  where the V_is, for $i = 1, \cdots, m$, are attributes, and each $v_i(o_1, \cdots, o_n, l_1, \cdots, l_k)$, for $i = 1, \cdots, m$, is a symbolic expression that defines a function $dom(O_1) \times \cdots \times dom(O_n) \times dom(L_1) \times \cdots \times dom(L_k) \rightarrow \Re$.

- A regression schema RS is a tuple

  $$< (L_1, f_1(O, \beta)), \cdots, (L_k, f_k(O, \beta)) >$$

  where each L_j, for $j = 1, \cdots, k$, is from the learning schema, and $f_i(O, \beta)$, for $i = 1, \cdots, k$, is a symbolic expression that defines a function in terms of the attributes $O =< O_1, \cdots, O_n >$ and parameters β.

In this paper we assume that all domains are $\Re$, including observation parameters, learning parameters, and views.

Definition 2 (*Regression Database Instance*). A regression database instance, $iRDB =< iOS, iLS, iRS >$, is composed of relation instances over an independent observation table schema, a learning table schema, and a regression model schema. An instance of the observation table schema, iOS, is a set of tuples of the form $< o_1, \cdots, o_n >$ where $o_i \in \text{dom}(O_i)$, $i \in 1 \cdots n$. An instance of the learning table schema, iLS, is a collection of tuples of the form $< o_1, \cdots, o_n, l_1, \cdots, l_k >$ where $o_i \in \text{dom}(O_i)$, for $i = 1, \cdots, n$, and $l_j \in \text{dom}(L_j)$, $j \in 1 \cdots k$. A regression instance, iRS, is of the form $< (L_1, \beta_1, \sigma_1), \cdots, (L_k, \beta_k, \sigma_k) >$, where β_i, σ_i for $i = 1, \cdots, k$, are the parameters for f_i in RS and the variance of the gaussian noise, correspondingly.

4 Query Language

This section defines the formal syntax and semantics for our reasoning query language as an extension to the relational algebra.

Definition 3 (*Condition*). A condition on a set of attributes *Attr* is defined recursively as follows:

- A *op* c is an *atomic condition*, where $A \in Attr$, c is a constant, and *op* is a relational comparison operator $(<, \leq, >, \geq, =)$;
- If C_1, C_2 and C_3 are conditions on $Attr$, then also $C_1 \wedge C_2$, $C_1 \vee C_2$, $\neg C_3$ are conditions on $Attr$.

Definition 4 (*Regular Condition*). A regular condition is a condition on attributes $< O_1, \cdots, O_n >$.

Definition 5 (*Random variable Condition*). A random variable condition is a condition on attributes $< L_1, \cdots, L_k >$ and $< V_1, \cdots, V_m >$.

Definition 6 (*Probability Condition*). A probability condition is an expression of the form $P(C_{rand}) \geq \delta$ or $P(C_{rand}) \leq \delta$, where P is for probability, C_{rand} is a random variable condition, and $\delta \in [0, 1]$.

Definition 7 (*Query Condition*). A query condition is defined recursively as follows:

- An atomic regular condition is a query condition;
- A probability condition is a query condition;
- If C_1, C_2, C_3 are query conditions, then also $C_1 \wedge C_2$, $C_1 \vee C_2$, $\neg C_3$ are query conditions.

Definition 8 (*Regression Database Query*). A regression database query is an expression of the form (π_x, σ_C), where $x \subseteq \{O_1, \cdots, O_n, L_1, \cdots, L_k, V_1, \cdots, V_m\}$, and C is a query condition.

Definition 9 (*Regression Query semantics*). A regression database query (π_x, σ_C) over a regression database instance I computes the following set of tuples: $\{(o_1, \cdots, o_n, \quad l_1, \cdots, l_k, \quad v_1, \cdots, v_m) \quad | \quad l_i \quad = f_i(o_1, \cdots, o_n, \beta) \quad \forall i \quad = \quad 1, \cdots, k, \quad v_i \quad = v_i(o_1, \cdots, o_n, l_1, \cdots, l_k)\ \forall j = 1, \cdots, m$, and $TV(C) =$ *true* $\}$, where $TV(C)$ is the truth value of the query condition C whose definition follows.

Definition 10 (*Truth value of a query condition*). The truth value of a query condition C is defined recursively as follows:

- If C is a regular condition, the truth value of C is the truth value of the regular condition, i.e.:
  - If $C \equiv (O\ op\ c)$, then $TV(C) = TV(O\ op\ c)$;
  - If $C \equiv (C_1 \wedge C_2)$, then $TV(C) = TV(C_1) \wedge TV(C_2)$, with C_1 and C_2 given regular conditions;
  - If $C \equiv (C_1 \vee C_2)$, then $TV(C) = TV(C_1) \vee TV(C_2)$, with C_1 and C_2 given regular conditions;
  - If $C \equiv \neg C_1$, then $TV(C) = \neg TV(C_1)$, with C_1 given regular condition.
- If C is a probability condition, the truth value of C is the truth value of the probability condition, i.e.:
  - $TV(P(C_{rand}) \geq \delta) =$ *True* if $P(C_{rand}) \geq \delta$ and *False* otherwise. (Similarly for $TV(P(C_{rand}) \leq \delta)$.) Here $P(C_{rand})$ is the probability of the event C_{rand} is under the independence assumptions of the random variables L_j, and using the distribution functions in the regression schema iRS.
- If C_1, C_2, C_3 are query conditions, $TV(C_1 \wedge C_2) = TV(C_1) \wedge TV(C_2)$, $TV(C_1 \vee C_2) = TV(C_1) \vee TV(C_2)$, and $TV(\neg C_3) = \neg TV(C_3)$.

It is important to note that, while an atomic random variable condition in queries only allows arithmetic comparisons of a learned or view attribute with a constant, it is possible to express an arithmetic comparison of two atributes, and, more generally, of two arbitrary linear combinations of attributes, i.e., $LC_1\ relop\ LC_2$, where LC_1 and LC_2 are two linear combinations of learned and view attributes, and $relop$ is an arithmetic comparison operator. This can be done by introducing a new view attribute V and defining it in the schema as $LC_1 - LC_2$, and then using the condition $V\ relop\ 0$.

5 Conceptual Query Evaluation

In this section, we show that REDB queries are computable by developing algorithms for conceptual query evaluation, and proving their termination and correctness. We allow view attributes that are defined in the REDB schema as arbitrary linear combinations of the learned attributes. Simpler algorithms for more restrictive view definition can be derived from the general ones. Probabilities are computed with finite precision ϵ; that is: when computing a value u, a value u' is returned such that $u - \epsilon \leq u' \leq u + \epsilon$. When evaluating the truth value of a condition $u\ relop\ c$, where u is a computed value, $relop$ is an arithmetic comparison operator, and c is a constant, we will return the truth value of $u'\ relop\ c$, where u' is a finite precision computation of u.

Algorithms 1 and 2 (Figures 1 and 2) compute a query and a query condition, respectively. Algorithm 1 is a simple iterative algorithm. It uses Algorithm 2 for Truth-Value evaluation of query conditions. Algorithm 2 recursively evaluates a query condition, which involves the computation of the probability of random variable conditions (with finite precision ϵ). This computation is given in Algorithm 3, and is based on computing what we call a *Primitive Disjunctive Normal Form*.

Definition 11 (*Primitive Disjunctive Normal Form*). We say that a condition F is in a *primitive disjunctive normal form* (PDNF) if it is of the form $\vee_i \wedge_j K_{ij}$, where K_{ij} is one of the following: $A < c_1$, $c_i \leq A < c_{i+1}$, $i = 1, \ldots, n-1$, or $A \geq c_n$, where $A \in Attr$ and $c_1 < c_2 < \cdots < c_n$ are all constants that appear in atomic conditions with A in F.

Algorihm 3 transforms a probabilistic condition into PDNF that may involve view attributes. The computation of PDNF conditions is given in Algorithm 4.

The main idea behind Algorithm 4 is that the disjuncts in the PDNF probabilistic conditions represent disjoint events. Thus, Algorithm 4 computes the probability of the PDNF probabilistic condition as the sum of probabilities of its disjunts. Each disjunct, being a conjunction of atomic conditions, is computed by Algorithm 5. Algorithm 5 computes the probability of a conjunction C of atomic con-

Algorithm 1
Input: Query (π_x, σ_C);
regression database instance $iRDB$; precision ϵ
Output: Query answer (as in Definition 9)
Method:
answer := $\emptyset$;
for each tuple $t = (o_1, \cdots, o_n)$ in iOS do
compute $TV(C, t, VS, RS, iRS, \epsilon)$ using Algorithm 2
if TV is true then
answer := answer $\cup\ \pi_x(o_1, \cdots, o_n,$
$l_1, \cdots, l_k, v_1, \cdots, v_k)$, where $l_1, \cdots, l_k$
are computed using $f_1, \cdots, f_k$ from iRS and
$v_1, \cdots, v_k$ are computed as defined in VS.

Figure 1. Conceptual Query Evaluation

ditions of the form $A\ relop\ c$, where A is a learned or a view attributes, and c is a constant. Because view attributes are defined as linear combinations of the learned attributes, we basically need to compute the probability of a condition that is a conjunction of linear inequalities over variables $L_1, \ldots, L_k$, which are the learned attributes. Geometrically C defines a polyhedral set S in k-dimensional space.

Assume first that S is bounded, and thus can be enclosed in the k-dimensional rectangle (bounded box) B. For this case, the key idea of Algorithm 5 is to partition B into smaller "boxes" by splitting each dimension into 2^i equal size intervals, for iteratively increasing i.

For each such partition, we can estimate the probability of the condition C by computing its lower and upper bounds. The lower bound is computed by considering all "boxes" inside B that are fully contained in S. The probability of the event represented by the union of these boxes is the sum of the probabilities of the boxes. Since each box is only defined using learned attributes/variables L_i, $i \in \{1, ..., k\}$, and they represent independent random variables, each "box" probability can be easily computed. Similarly, we compute the upper bound on the probability of C by considering all "boxes" in the partition that intersect C (not just contained in C). This process continues, until the lower and upper bound are close enough in terms of the required precision ϵ.

However, the set S that represents the condition C may not be bounded. For that reason, Algorithm 5 first artificially creates a k-dimensional rectangle ("box") B in such a way that the probability of being outside the box is bounded by $\epsilon/2$. Then, the algorithm proceeds to compute the probability of $S \cap B$ with finite precision $\epsilon/2$, so that the overall precision will be bounded by ϵ.

Clearly, Algorithm 1 (and all sub-algorithms) do terminate. By carefully looking at the construction in Algorithm 5 and at the selection of finite precision parameters in Algorithms 4 and 5, we can see that Algorithm 5 is also correct. We have the following theorem:

Algorithm 2
Input: Query condition C; tuple $(o_1, \cdots, o_n) \in iOS$;
VS, RS, iRS; precision ϵ
Output: True or false (as in Definition 10)
Method:
if C is an atomic regular condition then
return its truth value
else if C is a probability condition
$P(C_{rand}) \geq \delta$ (or $P(C_{rand}) \leq \delta$) then
compute $P(C_{rand})$ with precision ϵ using Algorithm 3
evaluate the truth value
$P(C_{rand}) \geq \delta$ (or $P(C_{rand}) \leq \delta$) resp.
return the evaluated truth value
else if $C = C_1 \wedge C_2$ then
return $TV(C_1) \wedge TV(C_2)$
else if $C = C_1 \vee C_2$ then
return $TV(C_1) \vee TV(C_2)$
else if $C = \neg C_1$ then
return $\neg TV(C_1)$

Figure 2. Computation of Truth Value

Theorem 1 Algorithm 1 for REDB query evaluation (1) terminates, and (2) is correct, i.e., it evaluates the correct answer (with an arbitrary small finite precision ϵ) according to Definition 9 of query semantics.

6 Conclusions and Future Work

In this paper we have introduced, formalized, and provided a conceptual computation framework for Regression Databases. Regression Databases are suitable for applications of reasoning about scenarios, events and courses of action, based on dynamically estimated characteristics and queries. We plan to pursue some related research questions, such as extending REDB to deal also with categorical domains; developing efficient algorithms for query processing and its optimization; implementing REDB; and extending the class of REDB queries.

References

[1] D. Barbarà, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):60–74, oct 1992.

[2] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. In *Proceedings of the 31st International Conference on*

Algorithm 3
Input: Random variable condition C_{rand};
tuple $(o_1, \cdots, o_n) \in iOS$; VS, RS, iRS;
precision ϵ
Output: $P(Crand)$ w/precision ϵ;
Method:
for each attribute $A \in \{L_1, \cdots, L_k, V_1, \cdots, V_m\}$ collect all constants that appear as atomic conditions with A in C_2 (Assume these constants are $c_1 \cdots c_l$);
for each atomic formula of the form A *op* c_i do
Case 1: $A \leq c_i$ or $A < c_i$
replace with $(A < c_1 \vee \cdots c_{i-1} \leq A < c_i)$
Case 2: $A \geq c_i$ or $A > c_i$
replace with ($c_1 \leq A < c_{i+1} \vee \cdots$
$c_{n-1} \leq A < c_n \vee \cdots c_n \leq A$)
assume the resulting formula is C_2;
transform C_2 into C_3 in DNF using the standard algorithm;
compute $P(C_3)$ with precision ϵ using Algorithm 4;
return result

Figure 3. Compute probability of random variable condition

Very Large Data Bases, pages 970–981. VLDB Endowment, 2005.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 551–562. ACM Press, 2003.

[4] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, 1996.

[5] D. Dey and S. Sarkar. PSQL: a query language for probabilistic relational data. *Data Knowl. Eng.*, 28(1):107–120, 1998.

[6] W.-C. Hou and Z. Zhang. Enhancing database correctness: a statistical approach. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 223–232, New York, NY, USA, 1995. ACM Press.

[7] W.-C. Hou, Z. Zhang, and N. Zhou. Statistical inference of unknown attribute values in databases. In *Proceedings of the Second International Conference on Information and Knowledge Management*, pages 21–30, New York, NY, USA, 1993. ACM Press.

[8] R. A. Johnson and D. W. Wichern, editors. *Applied Multivariate Statistical Analysis*. 3rd edition, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[9] G. Trajcevski. Probabilistic range queries in moving objects databases with uncertainty. In *Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 39–45, New York, NY, USA, 2003. ACM Press.

Algorithm 4
Input: C_{rand} in PDNF over attributes $< L_1, \cdots, L_k >$ and $< V_1, \cdots, V_m >$
with constants $c_1 < c_2 < \cdots < c_n$;
iRS; precision ϵ
Output: $P(Crand)$ w/precision ϵ;
Method:
assume $C_{rand} = \vee_{i=1}^{n} \wedge_{j=1}^{m} C_{ij}$, where C_{ij} is of the form $c_m < A_l \leq c_{m+1}$, $A_l \leq c_{m+1}$, or $c_m < A_l$, where $A \in < L_1, \cdots, L_k, V_1, \cdots, V_m >$ and $c_m, c_{m+1} \in (c_1 \cdots c_n)$;
return $\sum_i P(\wedge_j C_{ij})$ where $P(\wedge_j C_{ij})$ are computed for every i by Algorithm 5, with precision ϵ/N

Figure 4. Compute probability of PDNF

Algorithm 5
Input: C in PDNF over attributes $< L_1, \cdots, L_k >$ and $< V_1, \cdots, V_m >$;
iRS; precision ϵ
Output: $P(C)$ w/precision ϵ;
Method:
if C is unbounded find all $L_i s$ on which it is unbounded;
(Assume $L_{i_1}, \cdots, L_{i_l}$ are unbounded);
$C_1 := C \wedge \{\wedge_{j=1}^{l} (LB_{i_j} \leq L_{i_j} \leq UB_{i_j} \mid \forall (1 \leq j \leq l)$
$(P(L_{i_j} < LB_{i_j}) < \epsilon/4Nl$ and $P(L_{i_j} > UB_{i_j}) < \epsilon/4Nl)\}$
(C_1 is now bounded)

compute minimum bounding box for C_1,
$\wedge_{j=1}^{l} (LB_{i_j} \leq L_{i_j} \leq UB_{i_j})$;

i := 0;
repeat
i := i + 1;
split each dimension of the bounding box into 2^i intervals of equal size to get the partition $\{S_j\}_{j \in J}$, where $J = \{1, \ldots, 2^{ik}\}$;
find $\{S_j\}_{j \in J_1}$ that are contained in C_1 and $\{S_j\}_{j \in J_2}$ that intersect C_1;
$LB_i := \Sigma_{j \in J_1} P(S_j)$;
$UB_i := \Sigma_{j \in J_2} P(S_j)$;
until $UB_i - LB_i \leq \epsilon/2$
return $(LB_i + UB_i)/2$

Figure 5. Compute probability of a polyhedral set