Learning Multi-Relational Bayesian Networks with SQL all the way

ABSTRACT

The statistical analysis of structured data requires building structured machine learning models. We describe MRLBase, a new SQL-based framework that leverages the capabilities of an RDBMS to support multi-relational learning applications. Most previous machine learning applications assume a flat data representation with a single data table or matrix. However, many real-word datasets have a more complex structure, and are often maintained in a relational database. Building machine learning applications for multi-relational data requires new system capabilities. These capabilities include both representation and computation, such as: 1) A description language for specifying metainformation about structured random variables. 2) Efficient mechanisms for constructing, storing, and transforming complex statistical objects, such as cross-table sufficient statistics, parameter estimates, and model selection scores. 3) Computing model predictive scores for structured test instances. Our system design represents statistical objects as relational tables, on a par with the original data tables, so that SQL can be used to manage them. A case study on six benchmark databases shows how our system supports a challenging and important machine learning application, namely learning a Bayesian network model for an entire database. Our implementation shows how our SQL constructs in MRLBase facilitate fast, modular, and reliable program development. Empirical evidence indicates that leveraging the RDBMS capabilities achieves scalable learning and fast model testing.

1. BASELINES

how about moralizing, then compare with MLN structure learning.

Also, compare with our old code. I think we had the following limitations: only one relationship (two at most), and only link analysis on. Perhaps we could say that we overcome these limitations using SQL.

2. INTRODUCTION

Machine learning for large datasets is a growing application area at the intersection of machine learning and systems research. Several well-developed software packages implement standard machine learning algorithms (e.g., R, Weka). More recent developments support learning with large datasets. These packages assume that data is represented in a single table or data matrix, where each row represents a data point or feature vector. The single-table representation is appropriate when the data points represent a homogeneous class of entities with similar attributes, where the attributes of one entity are independent of those of others [**?**]. However, many real-word enterprise datasets have a more complex structure, with different classes of entities (customers, products, factories etc.), that have different attributes, and may be interrelated in multiple ways. Such heterogeneous data are often represented using a relational database management system (RDBMS). The field of *multi-relational learning* aims to extend machine learning to multi-relational data [**?**, **?**, **?**]. Database researchers have noted the usefulness of multi-relational statistical models for knowledge discovery representing uncertainty in databases [**?**, **?**, **?**]. Multi-relational learning is the process of building multi-relational statistical models.

In this paper we present MRLBase for "Multi-relational Learning Base", a framework for building the system capabilities required for multi-relational learning that go beyond what is required for single-table learning. Statistical system tasks include accessing data accesses, constructing, storing, querying, and transforming parameter estimates and model structures. MRLBase follows a client-server paradigm, where the client is a machine learning application for multi-relational data, and the server is an RDBMS that supports a machine learning application. The RDBMS is used not only to store data, but also to store structured objects for statistical analysis as first-class citizens in the database. The basic principle of MRLBase is to build the required system capabilities by leveraging RDBMS capabilities via SQL scripts, tables, and views. By separating system tasks from statistical issues, MRLBase facilitates extending single-table applications to multi-relational data. Our argument is that relational algebra can play the same role for multi-relational machine learning that linear algebra does for single-table machine learning: a unified language for both representing and computing with objects that support statistical analysis.

We provide an empirical evaluation of MRLBase on six benchmark databases, two of which contain over 1M records. MRLBase supports scalable multi-relational model learning, taking minutes on medium-size databases, and less than two hours on the largest database. Previously existing multi-relational learning methods do not scale to the largest database sizes. For the task of scoring model predictions on a set of test instances, the RDBMS capabilities easily implement block access to test instances, which leads to a 1,000 to 10,000-fold speed up compared to a simple loop.

Paper Organization. We begin with an overview of MRLBase. Based on this overview, we discuss the relationship to related works. The bulk of the paper discusses the details of implementing the system based on SQL: We begin with representing metainformation about relational random variables. Then we describe gathering multi-relational sufficient statistics via metaqueries. The sufficient statistics support the computation of model selection scores, and of model structure learning. These counting methods can be adapted for scoring models against structured test instances.

Contributions. The main contributions of this paper may be summarized as follows.

1. Identifying new system requirements for multi-relational machine learning that go beyond traditional single-table machine learning.
2. An integrated set of SQL-based solutions for providing these system capabilities, including
   - (a) Defining a default set of relational random variables, and extracting metainformation about them from the RDBMS system catalog.
   - (b) Computing contingency tables that store multi-relational sufficient statistics as database tables.
   - (c) Storing and scoring probabilistic models.

3. SYSTEM OVERVIEW

Figure 1 represents key system components. The starting point is a multi-relational database containing original data. We outline the main principles behind our design, then discuss how the key system components implement these principles.

3.1 Design Principles

The main design principles of MRLBase are the following.

(1) *Tabular Representation.* Structured objects for statistical analysis are stored in the relational database. The tabular representation makes it possible to use SQL high-level programming language for constructing and querying statistical objects.

(2) *Computation by SQL.* We use SQL queries to construct, query, and transform statistical objects. Server-side management of statistical objects reduces the computational resources required by the machine learning application.

(3) *Update by Views.* We create tables that represent statistical objects using the relational view mechanism. This minimizes the extent to which the machine learning application has to manage such updates.

(4) *Modularity and Independence.* We organize statistical objects in layers to minimize dependencies among them. Distributing the information about statistical objects across different database tables provides a more compact and more intuitive representation.

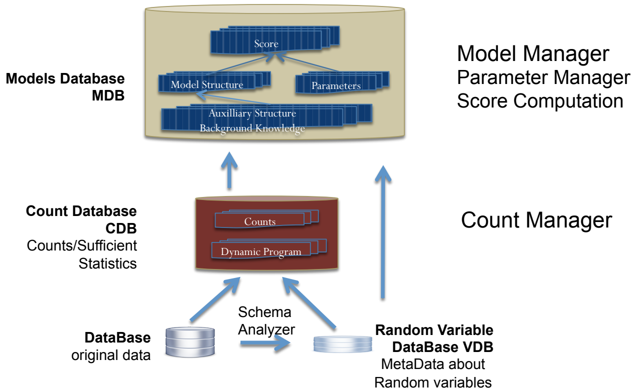


Figure 1: MRLBase Key Components. Arrows show dependencies.

3.2 System Components

3.2.1 The Schema Analyzer

Statistical analysis is based on a set of random variables. Single-table data is basically self-describing with respect to random variables: each column header other than row id fields represents a random variable. In contrast, a set of data tables that represents heterogeneous multi-relational data needs to be augmented with metadata about which columns represent which entity class. Such metadata requires a *data description language* [**?**]; in SQL the relevant key concepts are primary and foreign keys. A novel aspect of MRLBase is analyzing the RDBMS system catalog to translate metadata about primary and foreign keys into a specification of relational random variables.

The Schema Analyzer examines the information in the DB system catalog to define a default set of random variables for statistical analysis. Since the system catalog is itself stored in tables, the default set of random variables can be constructed using SQL queries. Metainformation about the random variables is stored in the **random variable database** *VDB*. This database may be edited by the user to add further random variables of interest. Another possibility is for a machine learning application to create the *VDB* database, for example based on metainformation specified in another format.

3.2.2 The Model Manager

The Model Manager supports the construction and querying of large structured statistical models. These models are also represented in relational database tables in the **Model Database** MDB. Services provided by the Model Manager include the following. (1) Compute parameter estimates for the model using the sufficient statistics in the Count Database. (2) Computing model characteristics such as the number of parameters or degrees of freedom in a model. (3) Computing a model selection score that quantifies how well the model fits the multi-relational data.

While MRLBase provides good solutions for each of these system capabilities in isolation, the ease with which the system components can be integrated is a key feature. Because information about random variables, sufficient statistics, and models is all represented in relational database tables, a machine learning application can access and combine the information in a uniform way via SQL queries.

4. RELATED WORK

We review the work around the topics of machine learning and data management most relevant to our research.

Single-Table Machine Learning and Data Management Systems. We briefly review several systems that leverage the advantages of advanced data management for machine learning applications. They still assume a single-table data representation of homogeneous data points. Most of this work complements ours, in that it focuses on different tasks such as inference or distributed processing. As a general comment, the MRLBase framework faciliates porting single-table methods to multi-relational data. Especially for single-table systems that leverage an RDBMS this is a natural extension that increases their usefulness even further.

There are several software collections that aim to provide users with high-level constructs for specifying statistical models and learning algorithms. These include the classic WinBUGS [?], as well as the more recent MADLib [?] and MLBase systems [?]. The MADLib vision is based on leveraging RDBMS capabilities through SQL programmin; MRLBase is a good fit for learning with a multi-relational component data source in the MADLib framework. The MLBase system emphasizes distributed processing and automatic refinement of machine learning algorithms and models. The MauveDB system [?] emphasizes the importance of several features for combining statistical analysis with databases. MauveDB presents model-based views of the *data* to the user, whereas MRLBase presents views of the models themselves to machine learning applications.

Multi-Relational Learning. Multi-Relational learning has been investigated by many researchers; most implemented systems use a logic-based representation of data derived from Prolog facts, that originated in the Inductive Logic Programming community; representative systems include Aleph and Alchemy [?]; for book-length overviews, please see [?, ?, ?, ?]. The logic-based approaches do not make use of SQL/RDBMS. Our case study using Bayesian network learning is largely focused on learning graphical models for multi-relational data.

Singh and Graepel [?] present an algorithm that translates key constraints from a relational database system catalog into a set of relational random variables and a Bayesian network structure. Differences include the following. (1) The Bayesian network structure is fixed and based on latent variables, rather than learned for observable variables only as in our case study. (2) The RDBMS is not used to support the learning after random variables have been extracted from the schema.

Computing sufficient statistics for single-table data has been well explored [?, ?], but much less for multi-relational statistics that combine information from different tables. Yin *et al.* [?] present a Virtual Join algorithm for computing sufficient multi-relational statistics. They do not use contingency database tables to store the sufficient statistics. In terms of our system, the Virtual Join algorithm is an alternative to metaqueries. Qian *et al.* [?] independently propose the use of contingency database tables. Their paper focuses on a Virtual Join algorithm for computing sufficient statistics that involve negated relationships. They do not discuss integrating contingency tables with other structured objects for multi-relational learning.

Multi-Relational Inference. Database researchers have developed powerful probabilistic inference algorithms for multi-relational models. These models leverage RDBMS capabilities for inference much as MRLBase does for learning. The BayesStore system [?] introduced the principle of treating all statistical objects as first-class citizens in a relational database as MRLBase does. The Tuffy system [?] achieves highly reliable and scalable inference for Markov Logic Networks (MLNs) with an RDBMS. The MRLBase can be used to learn an MLN. A very useful future project would be to combine MLN learning by MRLBase with inference by the Tuffy system to produce a single integrated RDBMS package for both learning and inference.

5. THE RANDOM VARIABLE DATABASE

Statistical analysis begins with a set of random variables. Formally, a **random variable** X is defined by a domain of possible values and a probability distribution over that domain. The more complex structure of multi-relational data leads to more complex structure for relational random variables, compared to random variables for single-table data. In this section we discuss what types of random variables are suitable for analyzing relational databases; we refer to these as *relational random variables.* Multi-relational learning requires making this structure explicit in machine-readable *metainformation.* We discuss how to find and store relevant metainformation about relational random variables. The metainformation for a random variable must include the following at a minimum. (1) The domain of the random variable. For discrete random variables, this is a finite set of possible values. (2) Pointers to the table and/or column in the original database that contains the data relevant to the random variables.

We adopt function-based notation from logic [?]. The expressive power of this formalism is equivalent to well-known logical query languages such as the domain relational calculus [?]. MRLBase can be adapted for other notational systems.

5.1 Relational Random Variables

A domain or **population** is a set of individuals. Individuals are denoted by lower case expressions (e.g., bob). A **functor** represents a mapping $f : \mathcal{P}_1, \ldots, \mathcal{P}_a \to V_f$ where f is the name of the functor, each $\mathcal{P}_i$ is a population, and V_f is the output type or **range** of the functor. In this paper we consider only functors with a finite range, disjoint from all populations. If $V_f = \{\mathrm{T}, \mathrm{F}\}$, the functor f is a (Boolean) **predicate**. A predicate with more than one argument is called a **relationship**; other functors are called **attributes**. We use uppercase for predicates and lowercase for other functors. Throughout this paper we assume that all relationships are binary, though this is not essential for our algorithm. A **Relational random variable** (RRV) is of the form $f(X_1, \ldots, X_a)$, where each X_i is a first-order variable [**?**, **?**]. Each first-order variable is associated with a population/type. In the context of RRVs, we therefore refer to first-order variables also as **population variables**. An RRV has two components: A functor and a list of population variables. We discuss first how the Schema Analzyer translates a relational database schema into a set of functors. Second, we describe a default method for combining population variables with functors.

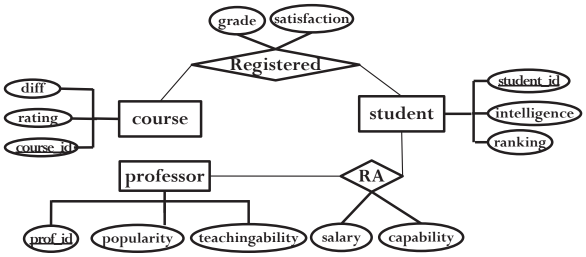
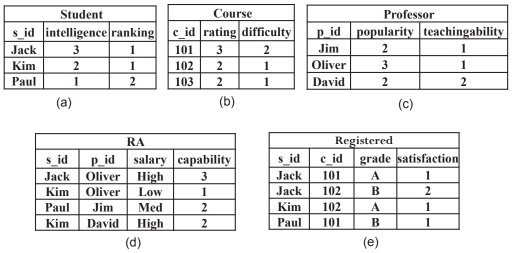


Figure 2: A relational ER Design for a university domain.



Student		
s_id	intelligence	ranking
Jack	3	1
Kim	2	1
Paul	1	2

(a)

Course		
c_id	rating	difficulty
101	3	2
102	2	1
103	2	1

(b)

Professor		
p_id	popularity	teachingability
Jim	2	1
Oliver	3	1
David	2	2

(c)

RA			
s_id	p_id	salary	capability
Jack	Oliver	High	3
Kim	Oliver	Low	1
Paul	Jim	Med	2
Kim	David	High	2

(d)

Registered			
s_id	c_id	grade	satisfaction
Jack	101	A	1
Jack	102	B	2
Kim	102	A	1
Paul	101	B	1

(e)

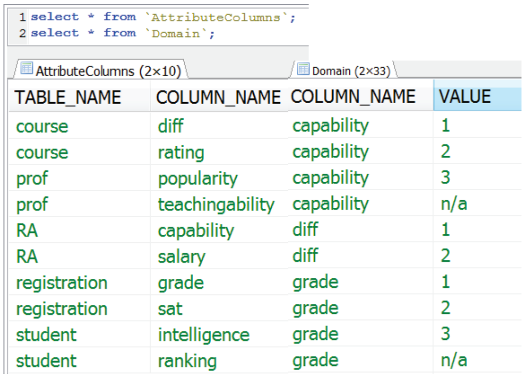
Figure 3: Database Table Instances: (a) *Student*, **(b)** *Course*, **(c)** *Professor*, **(d)** *RA*, **(e)** *Registered*.

5.2 Translating Entity-Relationship Models Into Functors

We assume a standard **relational schema** containing a set of tables, each with key fields, descriptive attributes, and possibly foreign key pointers. A **database instance** specifies the tuples contained in the tables of a given database schema. We assume that tables in the relational schema can be divided into *entity tables* and *relationship tables* (ER model) [**?**, Ch.2.2]. Figure 2 and Figure 3 show an ER diagram and an instance for a toy university domain, respectively. Our approach is to translate the components of the ER diagram into random variables for statistical analysis [**?**]. The translation of an ER diagram into a set of functors converts each element of the diagram into a functor, except for entity sets and key fields. Table 1 illustrates this translation.

ER Design	Type	Functor	*RRV*
Entity Tables	Population Variables	Student, Course	$\mathbb{S}, \mathbb{C}$
Relation Tables	Relationship	RA	RA($\mathbb{P}$,$\mathbb{S}$)
Entity Attributes	1Attributes	intelligence, ranking	{intelligence($\mathbb{S}$), ranking($\mathbb{S}$)} =1Attributes($\mathbb{S}$)
Relationship Attributes	2Attributes	capability, salary	{capability($\mathbb{P}, \mathbb{S}$), salary($\mathbb{P}, \mathbb{S}$)} = 2Attributes(RA($\mathbb{P}, \mathbb{S}$))

Table 1: Translation from ER Diagram to Relational Random Variable.



```
1 select * from `AttributeColumns`;
2 select * from `Domain`;
```

TABLE_NAME	COLUMN_NAME	COLUMN_NAME	VALUE
course	diff	capability	1
course	rating	capability	2
prof	popularity	capability	3
prof	teachingability	capability	n/a
RA	capability	diff	1
RA	salary	diff	2
registration	grade	grade	1
registration	sat	grade	2
student	intelligence	grade	3
student	ranking	grade	n/a

Figure 4: The metainformation about attributes represented in database tables. Left: The table *AttributeColumns* **specifies which tables and columns that contain the functor values observed in the data. The column name is also the functor ID. Right: The table** *Domain* **lists the domain for each functor.**

There are different types of functors corresponding to the different types of ER diagram components. The simplest type represents an attribute of an entity set. We refer to such functors as **1Attributes**. In Figure 2, there are six 1Attributes corresponding to attributes of Professors (2), Students (2), and Courses (2). The metainformation about 1Attributes can be stored in a database table as shown in Figure 4. 1Attributes correspond to *columns*. These are the only functors that appear in single-table data.

Relationship functors have the Boolean domain $\{\mathrm{T}, \mathrm{F}\}$. Relationship functors correspond to *tables*, not columns. There are two relationship variables in the diagram 2 corresponding to the *Registered* and *RA* relationships. A relationship table stores the information about which entities are related to each other in a certain way. For example, the database instance of Figure 3 represents that student Jack took course 101, and that student Kim did not take course 101. Including a relationship random variable in a statistical model allows the model to represent uncertainty about whether or not a relationship exists [**?**]. To relate a relationship variable to the original relationship data, we need to store pointers to the related entity sets as metainformation.

We refer to attributes of relationships as **2Attributes**. In Figure 2, there are four 2Attributes corresponding to attributes of Registered (2) and RA (2). An important issue for relational data is that the values of descriptive attributes of relationships are undefined for entities that are not related. Following [?], we represent this by introducing a new constant n/a in the domain of a 2Attribute; see Figure 4 (right). RRV's are called **1Variables** if their functor is a 1Attribute, **2Variables** if it is a 2Attribute, and **RVariables** if it is a relationship.

6. COUNT QUERIES AND MACHINE LEARNING TASKS

In terms of computational efficiency, the key issue for BN learning is computing event counts or sufficient statistics. Such counts are necessary for several machine learning tasks, such as:

- parameter estimation
- computing the BN likelihood function
- classification
- computing the pseudo-likelihood score of a candidate BN.

[elaborate]

Multi-relational data raise special challenges in computing sufficient statistics [copy from CIKM]. Our goal in this section is to show how SQL capabilities can be leveraged to address these challenges.

Sufficient statistics can be computed using an **SQL count query**, which has the form

```
%CREATE CT-table(<VARIABLE-LIST>) AS
SELECT COUNT(*) AS count, <VARIABLE-LIST>
FROM TABLE-LIST
GROUP BY VARIABLE-LIST
WHERE <Join-Conditions>, <restrictions>
```

6.1 Bayesian network multi-relational parameter estimation

6.2 Bayesian network multi-relational likelihood computation

6.3 Multi-relational Classification with Bayesian networks

6.4 Bayesian network multi-relational pseudo-likelihood computation

7. METAQUERIES: GENERATING DYNAMIC COUNT QUERIES

The **count query generation problem** is to generate a correct SQL count query for a given set of variables and restrictions that come from the machine learning client. [This needs to be done dynamically.] The query generation program should use metainformation about the random variables to automatically find the correct tables to join and the appropriate join conditions. Our solution is to stay within SQL and use an SQL **meta query** to generate the required SQL count query. Given a list of RRV's as input, the meta query is constructed as follows from the metainformation in the random variable DB.

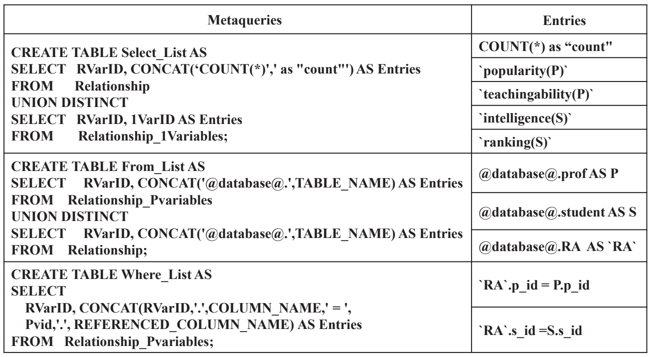
FROM LIST Find the tables referenced by the RRV's. An RRV references the entity tables associated with its population variables (see $VDB.Pvariables$). Relational RRV's also reference the associated relationship table (see $VDB.Relationship$).

WHERE LIST Add join conditions on the matching primary keys of the referenced tables in the WHERE clause. The primary key columns are recorded in table $VDB.KeyColumns$.

SELECT LIST For each attribute RRV, find the corresponding column name in the original database (see $VDB.AttributeColumns$). Rename the column with the ID of the RRV.

We represent a count-conjunction query of this form in four kinds of tables: the Select, From, Where and Group By tables. The Select table lists the entries in the Select clause of the target query, the From table lists the entries in the From clause, and similar for Where and GROUP BY tables. The entries of the Group By table are the same as in the Select table without the *count* column. Given the four query tables, the corresponding SQL count query can be easily executed in an application or stored procedure to construct the contingency table.

Figure 5 shows an example of metaqueries for the university database. This metaquery defines a view that in turn defines a contingency table for the random variable list associated with the relationship table RA. This list includes the 1Attributes of professors and of students, as well as the 2Attributes of the RA relationship. The resulting CT is like that of Figure **??**, but without the *Reg.* column and only with rows where the value of RA is true. This CT can extended to include counts for when RA is false using the Möbius Virtual Join [?].



Metaqueries	Entries

CREATE TABLE Select_List AS
SELECT RVarID, CONCAT('COUNT(*)',' as "count"') AS Entries
FROM Relationship
UNION DISTINCT
SELECT RVarID, 1VarID AS Entries
FROM Relationship_1Variables;
	COUNT(*) as "count"

	`popularity(P)`
	`teachingability(P)`
	`intelligence(S)`
	`ranking(S)`

CREATE TABLE From_List AS
SELECT RVarID, CONCAT('@database@.',TABLE_NAME) AS Entries
FROM Relationship_Pvariables
UNION DISTINCT
SELECT RVarID, CONCAT('@database@.',TABLE_NAME) AS Entries
FROM Relationship;
	@database@.prof AS P

	@database@.student AS S
	@database@.RA AS `RA`

CREATE TABLE Where_List AS
SELECT
RVarID, CONCAT(RVarID,'.',COLUMN_NAME,' = ',
Pvid,'.', REFERENCED_COLUMN_NAME) AS Entries
FROM Relationship_Pvariables;
	`RA`.p_id = P.p_id

	`RA`.s_id =S.s_id

Figure 5: Example of metaqueries and metaquery results based on university database. The parameter @*database*@ refers to the name of the input database.

8. BAYESIAN NETWORK MULTI-RELATIONAL STRUCTURE LEARNING

8.1 The Model Manager

8.2 Model Search

8.2.1 Model Selection Score

8.2.2 Model Optimization Search Algorithm

9. EVALUATION

9.1 Parameter Estimation

How many parameters, how long does it take. Baseline: big join table?

9.2 BN multi-relational likelihood computation

How long it takes for a single pass.

9.3 Classification

How long it takes for single object. Total time in loop (testing scenario).

9.4 Pseudo-likelihood

Compare with classification loop.

9.5 Model Learning

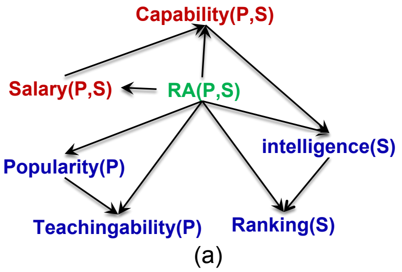
How long, how accurate. Don't use pseudo-likelihood because that's the evaluation metric for accuracy. Baseline RDN-Boost.

There is a large space of machine learning models, which require support for a diverse set of capabilities. We focus on the services that are required in almost all model selection tasks: 1) Estimating and storing parameter values. 2) Computing one or more model selection scores.

Our case study describes how MRLBase can be used to implement a challenging machine learning application: Constructing a Bayesian network model for a relational database. Managing Bayesian networks are a good illustration of typical challenges and how RDBMS capabilities can address them because: (1) Bayesian networks are a structured graphical model. (2) BN parameters are localized and not simply a flat vector. (3) Bayesian networks are widely regarded as a very useful model class in machine learning and AI, that supports decision making and reasoning under uncertainty. At the same time, they are considered challenging to learn from data. (4) Database researchers have proposed Bayesian networks for combining databases with uncertainty [**?**, **?**].

9.6 Bayesian Networks for Relational Data

A **Bayesian Network (BN)** is a directed acyclic graph (DAG) whose nodes comprise a set of random variables and conditional probability parameters. The parameters of the BN are the conditional probabilities of the form, $P(\mathit{child}|\mathit{parent\_values})$, that specify the probability of a child node value given an assignment of values to its parents. In this paper we consider only Bayesian networks whose nodes are relational random variables (called "Parametrized Bayesian Networks" in [**?**]). When discussing a BN, we interchangeable refer to its nodes or to its random variables. Figure 6 shows a Bayesian network for the University domain (only considering the *RA* relationship for simplicity.)



Capa(P,S)	RA(P,S)	Salary(P,S)	CP
n/a	F	n/a	1.00
4	T	high	0.45
5	T	high	0.36
3	T	high	0.18
3	T	low	0.20
2	T	low	0.40
1	T	low	0.40
2	T	med	0.22
3	T	med	0.44
1	T	med	0.33

(b)

Count	Capa(P,S)	RA(P,S)	Salary(P,S)
203	N/A	N/A	F
5	4	high	T
4	5	high	T
2	3	high	T
1	3	low	T
2	2	low	T
2	1	low	T
2	2	med	T
4	3	med	T
3	1	med	T

(c)

Figure 6: (a) Bayesian network for the University domain. (b) Conditional Probability table $\mathit{Capability}(\mathbb{P},\mathbb{S})\_\mathit{CPT}$, for the node $\mathit{Capability}(\mathbb{P},\mathbb{S})$. Only value combinations that occur in the data are shown. (c) Contingency Table $\mathit{Capability}(\mathbb{P},\mathbb{S})\_\mathit{CT}$ for the node $\mathit{Capability}(\mathbb{P},\mathbb{S})$ and its parents. Both CP and CT tables are stored as database tables.

$\mathit{BayesNet}(\underline{\mathit{child},\mathit{parent}})$
$@\mathit{nodeID}@\_\mathit{CPT}(\underline{@\mathit{nodeID}@,\mathit{parent}_1,\ldots,\mathit{parent}_k},\mathit{cp})$
$\mathit{Scores}(\underline{\mathit{nodeID}},\mathit{loglikelihood},\mathit{par},\mathit{AIC})$

Table 2: The main tables in the Models Database *MDB*. For a Bayesian network, the Models Database stores its structure, parameter estimates, and model selection scores. The @*nodeID*@ parameter refers to the ID of a Bayesian network nodes, for instance $\mathit{Capability}(\mathbb{P},\mathbb{S})$.

SQL Representation. A Bayesian network structure is a directed graph that can be stored straightforwardly in a database table *BayesNet* whose columns are *child* and *parent*. The table entries are the IDs of relational random variables defined in the Relational Random Variable database. An entry such as $(\mathit{Capability}(\mathbb{P},\mathbb{S}),\mathit{Salary}(\mathbb{P},\mathbb{S}))$ means that $\mathit{Capability}(\mathbb{P},\mathbb{S})$ is a child of $\mathit{Salary}(\mathbb{P},\mathbb{S})$. This table is stored in the Models Database *MDB*. The relational schema for the Models Database is shown in Table 2.

9.7 Parameter Manager

To make predictions with a model, we need to estimate values for its parameters. MRLBase stores the parameters for a Bayesian network as database tables, called **conditional probability tables.** Conditional probability tables have the same structure as contingency tables, but with a special column *cp* instead of *count*. Maximizing the data likelihood is the basic parameter estimation method for Bayesian networks. It can be shown that the maximum likelihood estimates use the observed frequency of a child value given its parent values [**?**, **?**].

SQL Construction of Conditional Probability Tables. Given the sufficient statistics in a contingency table, a conditional probability table containing the maximum likelihood estimates can be computed using SQL as follows. More complex smoothing methods such as the Laplace correction can be easily computed from the maximum likelihood estimates.

1. For each node, construct a local contingency table whose variable set comprises the node and its parents. This table can be computed from scratch using the count manager, or may already be available as part of model structure learning (see below).
2. Construct a parent contingency table whose variable set comprises the parents only: On the local contingency table, apply a query with SUM(count) AS parent_count in the Select clause and <parent-list> in the GROUP BY clause.
3. Carry out a natural join of the local contingency table with the parent contingency table, dividing each local contingency count by the parent_count. The natural join matches the values of parent variables.

All of these tables can be stored as views. Figure 6 shows a local contingency table and a conditional probability table for the node Capability(P, S).

9.8 Model Score Computation

Model structure learning uses a model selection score to find an optimum model for a given database. Model selection scores can be computed and stored in an RDBMS as well. Caching model selection scores is important for scalable structure learning [?]. A typical model selection approach is to maximize the likelihood of the data, balanced by a penalty term. For instance, the Akaike Information Criterion (AIC) is defined as follows [?].

$$AIC(G, \mathcal{D}) \equiv ln(P_{\widehat{G}}(\mathcal{D})) - par(G)$$

where $\widehat{G}$ is the BN G with its parameters instantiated to be the maximum likelihood estimates given the database $\mathcal{D}$, and $par(G)$ is the number of free parameters in the structure G. Computing this expression requires two terms, the likelihood and the number of parameters. The number of parameters for a node is the product the possible values for the parent nodes, multiplied by (the number of the possible values for the child node -1).

Assume that the maximum likelihood estimates are represented in a conditional probability table as discussed above. For a BN G, the log-likelihood function $ln(P_{\widehat{G}}(\mathcal{D}))$ can be computed node by node, as follows [?]. For each child node value, and for each combination of parent values: (1) find the instantiation count in the data for the conjunction of child node value and parent node values. (2) Find the conditional probability of the child node value given the parent node values. (3) Multiply the instantiation count by the logarithm of the conditional probability. Finally, sum these products to obtain a total likelihood score for the child node.

SQL Computation of Model Scores. We assume that for each node with ID @*nodeID*@, a conditional probability database table @*nodeID*@*_CPT* has been built in the Models database *MDB*. Similarly, we assume that a local contingency database table *CDB*.@*nodeID*@*_CT* has been built in the Count Database *CDB* (see Figure 6). The model likelihood for node @*nodeID*@ can be computed in SQL simply using the natural join of the two tables summing over a row-wise product, as follows. The *loglikelihood* value for @*nodeID*@ is inserted into the *Scores* table.

```
SELECT @nodeID@,  SUM
(MDB.@nodeID@_CPT.cp * CDB@nodeID@_CT.count)
AS loglikelihood
FROM MDB.@nodeID@_CPT NATURAL JOIN CDB.@nodeID@_CT
```

The complex aggregate computation in this short query illustrates how well the high-level SQL constructs support computation with structured objects. Computing the multi-relational likelihood in a general programming language (e.g., Java) would require significant development effort and result in a solution that is less concise, portable, and reliable.

To determine the number of parameters, the number of possible variable values can be found in the *Domain* table of the Random Variable Database *VDB*. The *par* number for @*nodeID*@ is also inserted into the *Scores* table. The AIC column is then defined as $AIC = loglikelihood - par$. These values can be inserted directly or the AIC column can be implemented as a derived column. Other model selection scores such as BIC and BDeu can be computed in a similar way given the model likelihood and number of parameters.

10. TEST SET PREDICTIONS

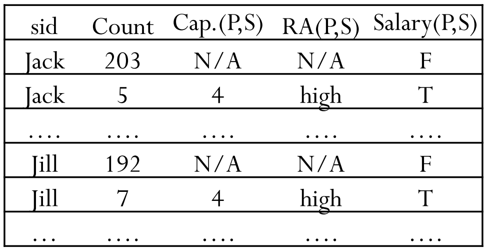
A basic procedure for evaluating the accuracy of a machine learning algorithm is the train-and-test paradigm, where the system is provided a training set for learning and then we test its predictions on unseen test cases. We first discuss how to compute a prediction for a single test case, then how to compute an overall prediction score for a set of test cases. For single-table data, BN prediction is straightforward. A test case is represented by a single predictive feature vector $\mathbf{x}$ and a class label y, and the BN product formula defines a joint probability for $P_G(\mathbf{x}, y)$ and hence a conditional probability $P(y|\mathbf{x})$. Thus for example, if we want to predict the intelligence of student Jack given that his ranking is 1, a single-table BN would define a conditional probability $P(\mathit{intelligence}(\mathit{jack}) = y | \mathit{rank}(\mathit{jack}) = \mathit{1})$ where y is a possible value for intelligence. For multi-relational data, a prediction model is much more difficult to specify, and a number of prediction models have been explored [?]. The difficulty is that a test case corresponds to a subdatabase or substructure, not just a single flat feature vector. For example, if we want to predict the intelligence of student Jack given information about his courses and his grades, we have to somehow aggregate the course information.

Log-linear models are a prominent prediction model class that has performed well with graphical models [?, ?, ?], including Bayesian networks [?]. The log-linear BN model can briefly be explained in terms of the model likelihood function $P_G(\mathcal{D})$ discussed in Section 8.8. Let Y denote a ground target node to be classified. The term “ground” refers to replacing population variables by target instances. For example, a ground target node may be *intelligence*$(jack)$. In this example, we refer to Jack as the **target entity**. Write $\mathbf{X}_{-Y}$ for a database instance that specifies the values of all ground nodes, except for the target node, which are used to predict the target node. Let $[\mathbf{X}_{-Y}, y]$ denote the completed database instance where the target node is assigned value y. The log-linear model uses the likelihood function as the joint probability of the label and the predictive features. In symbols, the log-linear model defines

$$P(y|\mathbf{X_{-Y}}) \propto P_G([\mathbf{X}_{-Y}, y]) \qquad (1)$$

where the model likelihoods of the possible class labels need to be normalized to define a conditional probabilities.

SQL Computation of the Log-Linear Classification Score. The obvious approach to computing the log-linear score would be to use the computation of Section 8.8. This is inefficient because instance counts that do not involve the target entity do not change the classification probability. For example, if Jack is the target entity, then the grades of Jill do not matter. This means that we need only consider query instantiations that match the appropriate population variable with the target entity (e.g., $\mathbb{S} = Jack$). For a given set of random variables, such query instantiation counts can be represented in a contingency table that we call the **target contingency table**. Figure 7 shows the format of a contingency table for target entities Jack and Jill.



sid	Count	Cap.(P,S)	RA(P,S)	Salary(P,S)
Jack	203	N/A	N/A	F
Jack	5	4	high	T
....	....	....	....	....
Jill	192	N/A	N/A	F
Jill	7	4	high	T
...	....	....	....	....

Figure 7: Target contingency tables for target = Jack and for target = Jill

Assuming that for each node with ID @*nodeID*@, a target contingency table *CDB*.@*nodeID*, *target*@*_CT* has been built in the Count Database *CDB*, the log-likelihood SQL is as in Section 8.8:

```
SELECT @nodeID@,  SUM
(MDB.@nodeID@_CPT.CP * CDB.@nodeID,target@_CT.count)
AS loglikelihood
FROM MDB.@nodeID@_CPT NAT. JOIN CDB.@nodeID,target@_CT
```

This query computes the classification score for a BN node, the total score is the sum over BN nodes. As is well-known in Bayes net theory, we need only sum scores for the target node and its children [?]. The new problem is finding the target contingency table. SQL allows us to solve this very easily by restricting counts to target entity in the WHERE clause. To illustrate, suppose we want to modify the contingency table query of Figure 5 to compute the contingency table for $\mathbb{S} = Jack$. We add the student id to the SELECT clause, and the join condition $S.s\_id = jack$ to the WHERE clause; see Table 3. (Omit apostrophes for readability.) The FROM clause is the same as in Figure 5. The metaquery of Figure 5 is easily changed to produce these SELECT and WHERE CLAUSES.

Next consider a setting where a model is to be scored against an entire test set. For concreteness, suppose the problem is to predict the intelligence of a set of students *intelligence*$(jack)$, *intelligence*$(jill)$, *intelligence*$(student_3), \ldots,$ *intelligence*$(student_m)$. An obvious approach is to loop through the set of test instances, repeating the likelihood query above for each single instance. Instead, SQL supports *block access* where we process the test instances as a block. Intuitively, instead of building a contingency table for each test instance, we build a single contingency table that stacks together the individual contingency tables (Figure 7). Blocked access can be implemented in a beautifully simple manner in SQL: we simply add the primary key id field for the target entity to the GROUP BY list; see Table 3.

In contrast, programming blocked access in a general purpose programming language would require significant development effort and probably new file or data structures. If the set of test instances is large, a stacked contingency table is also unlikely to fit into main memory. To sum up, the log-likelihood computation of Section 8.8 can be easily adapted to compute a log-linear classification score by adjusting the contingency table counts to the target entity. Only one or two small modifications to the log-likelihood SQL queries are required. This illustrates the modularity of MRLBase and the reusability of its components for different machine learning tasks.

11. EVALUATION

We describe the system and the datasets we used. Code was written in MySQL Script and Java, JRE 1.7.0. and executed with 8GB of RAM and a single Intel Core 2 QUAD Processor Q6700 with a clock speed of 2.66GHz (no hyper-threading). The operating system was Linux Centos 2.6.32. The MySQL Server version 5.5.34 was run with 8GB of RAM and a single core processor of 2.2GHz. All code and datasets are available on-line (pointer omitted for blind review).

11.1 Datasets

We used six benchmark real-world databases. For detailed descriptions and the sources of the databases, please see reference [?]. Table 4 summarizes basic information about the benchmark datasets. IMDB is the largest dataset in terms of number of total tuples (more than 1.3M tuples) and schema complexity. It combines the MovieLens database[1] with data from the Internet Movie Database (IMDB)[2] following [?].

For Bayesian network structure learning, we used MRLBase to implement the previously existing learn-and-join algorithm (LAJ). The LAJ method takes as input a joint contingency table for all relational random variables in the database, which we computed using the Count Manager.

[1] www.grouplens.org, 1M version
[2] www.imdb.com, July 2013

Table 3: SQL queries for computing target contingency tables supporting test set prediction. <Attribute-List> and <Key-Equality-List> are as in Figure 5.

Access	SELECT	WHERE	GROUP BY
Single	COUNT(*) AS count, <Attribute-List>, S.sid	<Key-Equality-List> AND S.s_id = jack	<Attribute-List>
Block	COUNT(*) AS count, <Attribute-List>, S.sid	<Key-Equality-List>	<Attribute-List>, S.sid

The model search strategy of the LAJ algorithm is an iterative deepening search for correlations among attributes along longer and longer chains of relationships. For more details please see [?].

Dataset	#Relationship Tables/ Total	#Self Relationships	#Tuples
Movielens	1 / 3	0	1,010,051
Mutagenesis	2 / 4	0	14,540
UW-CSE	2 / 4	**2**	712
Mondial	2 / 4	**1**	870
Hepatitis	3 / 7	0	12,927
IMDB	3 / 7	0	1,354,134

Table 4: Datasets characteristics. #Tuples = total number of tuples over all tables in the dataset.

Table 5 provides information about the number of relational random variables generated for each database, and the number of tuples required to store metainformation about them. More complex schemas and self-relationships lead to more random variables.

Dataset	# RRV	# Tuples in *VDB*
Movielens	7	72
Mutagenesis	11	124
UW-CSE	14	112
Mondial	18	141
Hepatitis	19	207
IMDB	17	195

Table 5: Random variable database statistics

The number of sufficient statistics reported in Table 6 is that for constructing the joint contingency table required for the Learn-and-Join algorithm. This number depends mainly on the number of random variables. The number of sufficient statistics can be quite large, over 15M for the largest dataset IMDB. RDBMS support is key for managing counts in such cases. Even with such large numbers, constructing contingency tables using the SQL metaqueries is feasible, taking just over 2 hours for the very large IMDB set. The number of Bayesian network parameters is much smaller than the number of sufficient statistics because it depends mainly on the indegree of the nodes. The Bayesian network can be seen as a compact representation of the statistical information in the joint contingency table [?]. So the difference between the number of parameters and the number of sufficient statistics measures how compactly the BN summarizes the statistical information in the data. Table 6 shows that Bayesian networks provide very compact summaries of the data statistics. For instance for the Hepatitis dataset, the ratio is 12,374,892/569 > 20,000. The IMDB database is an outlier, showing a complex correlation pattern that leads to a dense Bayesian network structure.

Dataset	# Database Tuples	# Sufficient Statistics (SS)	SS Computing Time (s)	#BN Parameters
Movielens	1,010,051	252	2.7	292
Mutagenesis	14,540	1,631	1.67	721
UW-CSE	712	2,828	3.84	241
Mondial	870	1,746,870	1,112.84	339
Hepatitis	12,927	12,374,892	3,536.76	569
IMDB	1,354,134	15,538,430	7,467.85	60,059

Table 6: Count Manager: Sufficient Statistics and Parameters

Table 8 shows that the graph structure of a Bayesian network contains a small number of edges relative to the number of parameters. The parameter manager provides fast maximum likelihood estimates for a given structure. This is because the variable set for a local contingency tables for BN parameter estimation comprises only a child node and its parents, so it is much smaller than the variable set in the joint contingency table.

Dataset	# Tuples in Bayes Net	# Bayes Net Parameters	Para. Learning Time (s)
Movielens	72	292	0.57
Mutagenesis	124	721	0.98
UW-CSE	112	241	1.14
Mondial	141	339	60.55
Hepatitis	207	569	429.15
IMDB	195	60,059	505.61

Table 7: Model Manager Evaluation.

Figure 8 compares computing predictions on a test set using an instance-by-instance loop, with a separate SQL query for each instance, vs. a single SQL query for all test instances as a block. The blocked access method is 1,000-10,000 faster.

Dataset	# Tuples in Bayes Net	# Bayes Net Parameters	Para. Learning Time (s)
Movielens	72	292	0.57
Mutagenesis	124	721	0.98
UW-CSE	112	241	1.14
Mondial	141	339	60.55
Hepatitis	207	569	429.15
IMDB	195	60,059	505.61

Table 8: Model Manager Evaluation.

Table 9 reports result for the complete learning of a Bayesian network, structure and parameters. It benchmarks MRL-Base against functional gradient boosting, a state-of-the-art

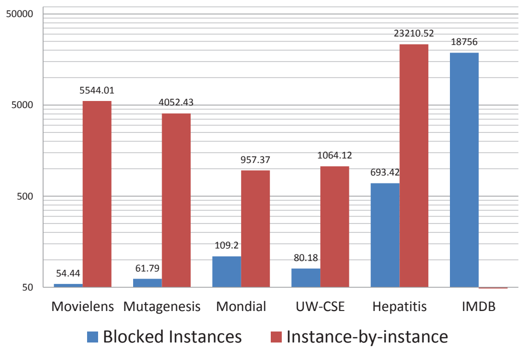


Figure 8: Times (s) for Computing Predictions on Test Instances. The right red column shows the time for looping over single instances using the Single Access Query of Table 3. The left blue column shows the time for the Blocked Access Query of Table 3.

multi-relational learning approach that construct a graphical model with parameter estimates[**?**]. MLN_Boost learns a Markov Logic Network, and RDN_Boost a Relational Dependency Network. We used the Boostr implementation [**?**]. To make the results easier to compare across databases and systems, we divide the total running time by the number of random variables for the database (Table 5). Table 9 shows that structure learning with MRLBase is fast: even the large complex database IMDB requires only around 8 minutes/node. Compared to the boosting methods, MRLBase shows excellent scalability: the competitor methods do not terminate on IMDB database, and while RDN_Boost terminates on the MovieLens database, it is almost 5,000 times slower than MRLBase. While the Learn-and-Join algorithm is more efficient than the boosting ensemble search, much of its speed is due to quick computation of sufficient statistics. As the last column of Table 9 shows, on the larger datasets MRLBase spends about 80% of computation time on gathering sufficient statistics. This suggests that a large speedup for the boosting algorithms could be achieved if they used the MRLBase approach.

We do not report accuracy results due to space constraints and because predictive accuracy is not the focus of this paper. On the standard conditional log-likelihood metric, as defined by Equation 1, the BN learned by MRLBase performs better than the boosting methods on all databases. This is consistent with the results of previous studies [**?**].

Dataset	RDN_Boost	MLN_Boost	MRLBase	MRLBase-CT
MovieLens	92.7min	N/T	1.12	0.39
Mutagenesis	118	49	1	0.15
UW-CSE	15	19	1	0.27
Mondial	27	42	102	61.82
Hepatitis	251	230	286	186.15
IMDB	N/T	N/T	524.25	439.29

Table 9: Learning Time Comparison with other multi-relational learning systems. Unless otherwise noted, times are in seconds.

Conclusion. MRLBase leverages RDBMS capabilities for scalable management of statistical analysis objects. It efficiently constructs and stores large numbers of sufficient statistics and parameter estimates. The RDBMS support for multi-relational learning translates into orders of magnitude improvements in speed and scalability.

12. CONCLUSION AND FUTURE WORK

Compared to traditional learning with a single data table, learning for multi-relational data requires new system capabilities. In this paper we described MRLBase, a system that leverages the existing capabilities of an SQL-based RDBMS to support multi-relational learning. Representational tasks include specifying metainformation about structured random variables that are appropriate for multi-relational data, and storing the structure of a learned model. Computational tasks include storing and constructing sufficient statistics (event counts), and computing parameter estimates and model selection scores based on the sufficient statistics. We showed that SQL scripts can be used to implement these capabilities, with multiple advantages. These advantages include: 1) Fast program development through high-level SQL constructs for complex table and count operations. 2) Managing large and complex statistical objects that are too big to fit in main memory. For instance, some of our benchmark databases require storing and querying millions of sufficient statistics. Empirical evaluation on six benchmark databases showed significant scalability advantages from utilizing the RDBMS capabilities: Both structure and parameter learning scaled well to millions of data records, beyond what previous multi-relational learning systems can achieve.

Future Work. On the RDBMS side, our implementation has used simple SQL plus indexes. Further optimizations are likely possible, especially for view materialization and the key scalability bottleneck of computing multi-relational sufficient statistics. An important direction is to integrate MRLBase with probabilistic inference systems that also utilize RDBMS capabilities, such as BayesStore and Tuffy. Further potential application areas for MRLBase include managing massive numbers of aggregate features for classification [**?**], and collective matrix factorization [**?**].

In sum, we believe that the succesful use of SQL presented in this paper shows that relational algebra can play the same role for multi-relational learning as linear algebra for single-table learning: a unified language for both representing statistical objects and for computing with them.