Mach Learn (2015) 100:217–254
DOI 10.1007/s10994-015-5483-2

Learning relational dependency networks in hybrid domains

Irma Ravkic[1] · **Jan Ramon**[1] · **Jesse Davis**[1]

Received: 27 April 2014 / Accepted: 22 January 2015 / Published online: 5 May 2015

Abstract Statistical relational learning (SRL) is concerned with developing formalisms for representing and learning from data that exhibit both uncertainty and complex, relational structure. Most of the work in SRL has focused on modeling and learning from data that only contain discrete variables. As many important problems are characterized by the presence of both continuous and discrete variables, there has been a growing interest in developing hybrid SRL formalisms. Most of these formalisms focus on reasoning and representational issues and, in some cases, parameter learning. What has received little attention is learning the structure of a hybrid SRL model from data. In this paper, we fill that gap and make the following contributions. First, we propose hybrid relational dependency networks (HRDNs), an extension to relational dependency networks that are able to model continuous variables. Second, we propose an algorithm for learning both the structure and parameters of an HRDN from data. Third, we provide an empirical evaluation that demonstrates that explicitly modeling continuous variables results in more accurate learned models than discretizing them prior to learning.

Keywords Statistical relational learning · Hybrid domains · Structure learning

Editors: Toon Calders, Rosa Meo, Floriana Esposito, and Eyke Hüllermeier.

✉ Irma Ravkic
Irma.Ravkic@cs.kuleuven.be

Jan Ramon
Jan.Ramon@cs.kuleuven.be

Jesse Davis
Jesse.Davis@cs.kuleuven.be

[1] Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

1 Introduction

Statistical relational learning (SRL) (Getoor and Taskar 2007) studies formalisms that combine relational representations such as first-order logic with models for capturing uncertainty. The motivation underlying SRL is that real-world domains such as patient clinical histories, molecular structures, and social networks are characterized by the presence of data that are complex, highly structured and uncertain. Many real-world problems are also hybrid in that they contain both discrete and continuous variables. Examples of such domains include robotics, where a robot's location is described by continuous variables and properties of encountered objects can be described by discrete variables; clinical histories, where a patient's temperature and blood pressure represent continuous variables while their gender and diagnoses are discrete variables; and biology, where spatial relationships between molecules are modelled as continuous variables and atom types and amino acid types are discrete properties. Unfortunately, few formalisms can cope with structured and uncertain data that contain both continuous and discrete variables. On the one hand, hybrid Bayesian networks (Murphy 1998) model uncertainty for both continuous and discrete variables, but not relations. On the other hand, SRL approaches such as logical Bayesian networks (LBNs) (Fierens et al. 2005), probabilistic relational models (Getoor et al. 2001), and relational dependency networks (Neville and Jensen 2007) capture both structure and uncertainty in problems but are generally restricted to discrete data.

To address this shortcoming, there has recently been increased interest in designing hybrid SRL formalisms such as Hybrid Markov Logic Networks (HMLNs) (Wang and Domingos 2008), Hybrid ProbLog (HProbLog) (Gutmann et al. 2011), Continuous Bayesian Logic Programs (CBLPs) (Kersting and De Raedt 2001), Learning Modulo Theories (LMT) (Teso et al. 2013) and Hybrid Probabilistic Relational Models (HPRMs) (Narman et al. 2010). The vast majority of the work on hybrid SRL has focused on two issues. The first is building up the machinery needed to represent continuous variables within the various SRL formalisms. The second is adapting inference procedures such that they work for hybrid domains. Some formalisms provide support for learning the parameters of a handcrafted structure from data. What has received little attention to date is designing algorithms that are able to learn the structure of a hybrid SRL model (i.e., the dependencies among the variables and relations in a domain) from data.

In this paper we fill that gap by exploring structure learning within a hybrid SRL context. First, we describe hybrid relational dependency networks (HRDNs) a novel formalism which extends RDNs to handle continuous variables. HRDNs approximate a joint probability distribution with a set of conditional probability distributions (CPDs). We discuss several local conditional probability distributions that are adept at modeling continuous variables. Second, we present an algorithm that is able to learn the structure of an HRDN from data.[1] To the best of our knowledge, this is the first attempt to perform structure learning in the hybrid SRL setting. Third, we empirically evaluate our proposed algorithm on one synthetic and one real-world data set. We find that applying our approach to the original hybrid data results in more accurate learned models than discretizing the data prior to learning.

[1] A publicly available implementation of our algorithm can be found on http://dtai.cs.kuleuven.be/ml/systems/llm.

2 Background

We will review both propositional and relational dependency networks. First, we will introduce some general definitions and notational conventions used throughout the paper.

We consider two types of variables. First, a random variable, called a *randvar*, is a variable that has an associated range of values it can take based on a probability distribution. Second, a logical variable, called a *logvar*, is a variable that has a finite domain of possible values it can take. A logvar is a placeholder for objects such as students or courses. We denote variables with uppercase letters and specific values with lowercase letters. Given a set of variables $\mathcal{X}$, a boldface lowercase letter, such as $\mathbf{x}$, represents an assignment of a value to each variable in the set.

2.1 Propositional dependency networks

A dependency network (DN) (Heckerman et al. 2001) is a (cyclic) directed probabilistic model that approximates a joint probability distribution over a set of random variables with a set of conditional probability distributions (CPDs). A DN is a tuple $(\mathcal{X}, dep)$ where $\mathcal{X}$ is a set of randvars and dep is a function that maps each randvar $X \in \mathcal{X}$ to a conditional probability distribution $p(X \mid Parents(X))$, where $Parents(X) \subseteq \mathcal{X} \setminus \{X\}$. The CPD quantifies how X depends on the variables in $Parents(X)$. A DN can be represented visually as a directed graph $G = (V, E)$, containing one vertex V_X for each randvar $X \in \mathcal{X}$ and a directed arc from vertex V_X to vertex V_Y iff $X \in Parents(Y)$.

Learning the structure of a DN from data requires determining $Parents(X)$ for each $X \in \mathcal{X}$ (i.e., the dependency structure) and the parameters of the CPD for X. Even though the parameters of CPDs can be estimated by using a variety of regression or classification techniques, the standard method is to use probabilistic decision trees. One scoring function that is often used when learning DNs (and other probabilistic graphical models) is *pseudo-loglikelihood* (PLL) (Besag 1974). Optimizing the PLL has the advantages that it can be decomposed into maximizing the log-likelihood for each variable independently and calculating it does not require computing the partition function (that is, summing over all possible configurations of the randvars). The PLL of an assignment $\mathbf{x}$ to randvars $\mathcal{X}$ of a DN is calculated as:

$$PLL(\mathbf{x}) = \sum_{i=1}^{n} log\,[p(X_i = x_i | Parents(X_i))]. \tag{1}$$

Learning each CPD independently could result in an inconsistent model. That is, there may be no joint probability distribution such that it is possible to apply the rules of probability to the joint distribution in order to derive each learned CPD.

Regardless of whether a DN is consistent, applying an *ordered* Gibbs sampler to the DN's CPDs results in a unique distribution, given that each variable in the DN is discrete and each CPD in the DN is positive (Heckerman et al. 2001). Ordered Gibbs sampling randomly selects the initial value for each random variable, and then in each Gibbs sweep iterates over the variables in a fixed order and resamples the value of each X_i from its local distribution $p(X_i | Parents(X_i))$. If the DN is consistent, it generates the joint probability distribution. If the DN is inconsistent, this procedure is called an *ordered pseudo-Gibbs* sampler (Heckerman et al. 2001).

2.2 Relational dependency networks

Next, we review relational dependency networks (RDNs) (Neville and Jensen 2007). There are several ways to define RDNs, but we use a definition that uses first-order logic as a template language for constructing propositional dependency networks. We first briefly review the relevant concepts from first-order logic, then we define RDNs. Throughout the discussion, we will use a slightly modified version of the popular *university* model (Getoor et al. 2001) as a running example.

We use the datalog subset of first-order logic. The alphabet consists of three types of symbols: constants, logical variables, and predicates. A *constant* represents a specific object and is denoted with a lower-case letter (e.g., `pete`). A *logical variable* (*logvar*) `X` is a variable ranging over the objects in the domain. Logical variables may be *typed* in which case they represent placeholders for a specific subset of objects in the domain. *Predicate* symbols `P`/n, where $n \geq 0$ is the arity of the predicate, represent properties of objects or relations among objects. We use a typed language, that is, every argument position of a predicate has a type. Each predicate `P` has a finite range, denoted $range(\texttt{P})$. In contrast to traditional logic, we do not restrict the range of a predicate to $\{false, true\}$. For example, the range of a student's intelligence could be $\{\texttt{low}, \texttt{med}, \texttt{high}\}$. An *atom* is of the form $\texttt{P}(\texttt{t}_1, \ldots, \texttt{t}_n)$ where `P`/`n` is a predicate and each $\texttt{t}_i$ is an object or a logvar. The range of an atom is the range of its predicate. A literal is an atom or its negation. An atom is *ground* if all its arguments are constants. A *substitution*, denoted $\{\texttt{X}_1/\texttt{t}_1, \ldots, \texttt{X}_n/\texttt{t}_n\}$, maps each logvar $\texttt{X}_i$ to $\texttt{t}_i$, where $\texttt{t}_i$ is a logvar or a constant. A *grounding substitution* θ for an expression (e.g., an atom or a set of logvars) maps each logvar occurring in that expression to a constant. The set of all grounding substitutions for an expression `E` is denoted $grsub(\texttt{E})$. The result of applying a substitution to an atom `a` is denoted $\texttt{a}\theta$.

Similar to LBNs (Fierens et al. 2005), we use a set of statements to define the random variables in a domain:

$$random(\texttt{H}) \leftarrow l_1, \ldots, l_n$$

where `H` is an atom, and $l_1, \ldots, l_n$ is a conjunction of literals. Given a set of random variable declarations RVD, the set of random variables Φ is the set of all ground atoms $\texttt{A}\theta$ for which there is a random variable declaration $random(\texttt{A}) \leftarrow l_1, \ldots, l_n$ in RVD and a substitution θ such that $l_1\theta$, …, $l_n\theta$ is true given the background knowledge (amongst others specifying which ground atoms of the predicates in the body of the random variable declaration rules are true). For example, the random variable declaration for the atom `takes(S,C)`

$$random(\texttt{takes(S,C)}) \leftarrow \texttt{student(S), course(C)} \tag{2}$$

creates one randvar for each student `S` and course `C` in the domain.

It must always be possible to evaluate the conjunction in the right-hand side of a random variable declaration, and we will use a closed-world assumption to guarantee this. As is common practice in many other probabilistic logical model frameworks (Fierens et al. 2005; Richardson and Domingos 2006; Getoor et al. 2001), our random variable declarations specify all random variables that are potentially of interest. For example, the random variable declaration

$$random(\texttt{grade(S,C)}) \leftarrow \texttt{student(S),course(C)} \tag{3}$$

specifies that every student gets a grade for every course, even though a precondition for obtaining a grade is that student `S` must take course `C`. In this case, `grade(S,C)` would have a special value *not_relevant* in its domain, and we would have the background knowledge

$$\texttt{grade(S,C)} = not\_relevant \Leftrightarrow \texttt{takes(S,C)} = false \quad (4)$$

We refer to these statements as *relevancy conditions*. Later, when learning the conditional dependency for `grade(S,C)` on `takes(S,C)` and other random variables, we can easily use such hard background knowledge and reduce the learning problem to the subspace of the values of the parent random variables for which the dependent random variable is relevant.

Let $\texttt{h}\theta$ be a random variable. Given background knowledge, an interpretation I assigns a value to $\texttt{h}\theta$ from its range or it assigns the special value $not\_relevant$ iff there exists a relevancy condition $\texttt{h}\theta \Leftrightarrow \varphi$ in the background knowledge and $\varphi\theta$ is true in I. The set of all groundings of a predicate `P` that have an assigned value $v \neq not\_relevant$ in interpretation I is denoted as $gr(\texttt{P})^I$. We refer to the randvars in $gr(\texttt{P})^I$ as `P`'s *relevant* randvars.

Now, we will introduce relational features. For this, we first need to define aggregation functions.

Definition 1 (*Aggregation function*) An aggregation function for a domain D is a function that maps every finite multiset of elements from D to a single value from a range R.

For example, *mode* is an aggregation function that maps a multiset of values from D to the most frequently occurring value in the multiset.

Definition 2 (*Discrete relational feature*) Let L be a set of logvars, C be a conjunction of randvar-value tests of the form $G = v$ where G is an atom and $v \in range(G)$, A be an atom, and α be an aggregation function taking as input multisets of elements of $range(A)$. Assume the ranges of A, all atoms in C and α are discrete. Then, a *discrete relational feature* $\mathcal{F}_{L:C,A,\alpha}$ is a function that maps any $\theta \in grsub(L)$ and interpretation I to

$$\mathcal{F}_{L:C,A,\alpha}(\theta, I) = \alpha\left(\{I(A\theta\theta') \mid \theta' \in grsub(A\theta, C\theta)\ and\ C\theta\theta'\ holds\ in\ I\}\right)$$

where we say $C\theta\theta'$ holds in I iff $\forall(G = v) \in C, I(G\theta\theta') = v$.

A feature's range is the range of its aggregation function α. The length of a feature is equal to the number of randvar-value tests in C plus one (for A).

There are two cases for grounding a relational feature that warrant mention:

(a) $|\{I(A\theta\theta') \mid \theta' \in grsub(A\theta, C\theta)\ and\ C\theta\theta'\ holds\ in\ I\}| = 1$, for all $\theta \in grsub(L)$
(b) $|\{I(A\theta\theta') \mid \theta' \in grsub(A\theta, C\theta)\ and\ C\theta\theta'\ holds\ in\ I\}| = 0$, for all $\theta \in grsub(L)$

The first case uses $value$ to denote the identity function which returns $I(A\theta\theta')$. For example, if each student `S` has exactly one value for intelligence, then the relational feature $\mathcal{F}_{\{\texttt{S}\}:\emptyset,\texttt{intelligence(S)},value}$ simply returns the value taken by the randvar `intelligence(S)`, which represents the `intelligence` of a student `S`, in interpretation I. The second case requires applying an aggregation function to the empty set. Some aggregation functions (e.g., mode) are not defined on the empty set, and in this case $\mathcal{F}_{L:C,A,\alpha}(\theta, I)$ returns the value $undefined$.

Example 1 Consider the following relational feature:

$$\mathcal{F}_{\{\texttt{S}\}:\texttt{grade(S,C)=low},\texttt{difficulty(C)},mode}$$

where `C` is a logvar denoting courses and `S` is a logvar denoting students. This feature calculates the mode of the difficulties for the courses where a student received a low grade. If a student has taken no courses or received no low grades, then, as discussed above, *mode* would return the value *undefined*.

Definition 3 (*Discrete dependency statement*) A *discrete dependency statement* is of the form $\mathtt{G} \mid \text{Parents}(\mathtt{G})$. $\mathtt{G}$ is the *target* atom that has a discrete range and whose arguments are all logvars. Parents($\mathtt{G}$) is a set of discrete relational features, where for each $\mathcal{F}_{L:C,A,\alpha} \in$ Parents($\mathtt{G}$), L is a subset of the logvars in $\mathtt{G}$. Each dependency statement has an associated conditional probability distribution (CPD) which quantifies how the target atom depends on its parent set.

Example 2 An example of a discrete dependency statement is:

$$\mathtt{intelligence(S)} \mid \mathcal{F}_{\{\mathtt{S}\}:\mathtt{takes(S,C)=true},\mathtt{grade(S,C)},mode}$$

which states that a student's intelligence depends on the *mode* of grades received across all courses the student has taken. As each student can take a varying number of courses, an aggregation function, such as *mode* in this example, is needed to combine the values from the varying number of parents into a single value.

We are now ready to formally define an RDN:

Definition 4 (*RDN*) An RDN is a tuple $(\mathcal{P}, RVD, dep)$, where $\mathcal{P}$ is a set of predicates, each with a discrete range, RVD is a set of randvar declarations, and dep is a function that maps each $\mathtt{P} \in \mathcal{P}$ to a discrete dependency statement.

An RDN $(\mathcal{P}, RVD, dep)$ is a template for constructing propositional DNs. Given the background knowledge and a set of randvar declarations RVD, an induced DN has a node for each randvar $\mathtt{G}\theta \in \Phi$.

The parent set of a ground atom $\mathtt{G}\theta$ in a dependency network is defined as

$$Parents(\mathtt{G}\theta) = Parents_A(\mathtt{G}\theta) \cup Parents_C(\mathtt{G}\theta)$$

where

$$\begin{aligned} Parents_A(\mathtt{G}\theta) &= \left\{A\theta\theta' \mid \exists \mathcal{F}_{L:C,A,\alpha} \in Parents(\mathtt{G}) : \theta' \in grsub((C\theta, A\theta))\right\} \\ Parents_C(\mathtt{G}\theta) &= \cup\left\{C\theta\theta' \mid \exists \mathcal{F}_{L:C,A,\alpha} \in Parents(\mathtt{G}) : \theta' \in grsub((C\theta, A\theta))\right\} \end{aligned} \quad (5)$$

There is an arc between two ground atoms $\mathtt{G}\theta$ and $\mathtt{G}'\theta$, if $\mathtt{G}'\theta \in Parents(\mathtt{G}\theta)$. The CPDs are shared across all randvars that originate from the same predicate.

The pseudo-loglikelihood of an RDN M for an interpretation I involves only the relevant randvars and it is calculated as:

$$PLL(M; I) = \sum_{\mathtt{P} \in \mathcal{P}} \sum_{g \in gr(\mathtt{P})^I} \log\ [p(I(\mathtt{g}) \mid I(Parents(\mathtt{g}))]. \quad (6)$$

Example 3 Consider the following simple RDN for a domain with the following randvar declarations:

$$\begin{aligned} &\mathtt{random(intelligence(S))} \leftarrow \mathtt{student(S)} \\ &\mathtt{random(takes(S,C))} \leftarrow \mathtt{student(S)}, \mathtt{course(C)} \\ &\mathtt{random(grade(S,C))} \leftarrow \mathtt{student(S)}, \mathtt{course(C)} \\ &\mathtt{random(difficulty(C))} \leftarrow \mathtt{course(C)} \end{aligned}$$

where each predicate has a discrete range and the following dependency statement:

$$\mathtt{grade(S,C)} \;\Big|\; \begin{array}{l} \mathcal{F}_{\{\mathtt{S}\}:\emptyset,\mathtt{intelligence(S)},\mathtt{value}}, \\ \mathcal{F}_{\{\mathtt{C}\}:\emptyset,\mathtt{difficulty(C)},\mathtt{value}} \end{array}$$

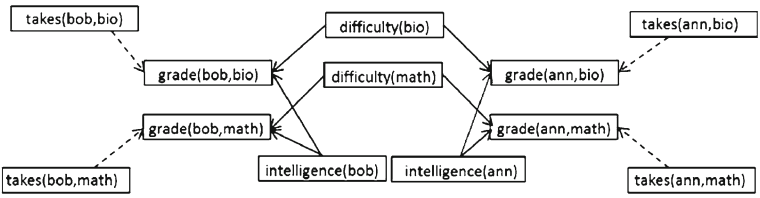

Fig. 1 The DN induced by grounding the RDN specified in Example 3. The *dashed arrows* specify the relevancy condition on `grade/2`

The dependency states that a student's grade in a course depends on the student's intelligence and the difficulty of the course. Note that this statement says that all ways of instantiating the logvars `S` and `C` have an identical probabilistic relationship with `S`'s intelligence and `C`'s difficulty. Figure 1 shows an induced propositional DN for this RDN given the relevancy condition on `grade/2` specified in (4), and a domain with two students `bob` and `ann`, and two courses `math` and `bio` (short for biology). The dashed arrows denote the relevancy conditions for the `grade/2` randvars.

Given that RDNs are templates for constructing DNs, they inherent the semantics of DNs (Neville and Jensen 2007). Namely, a consistent RDN specifies a joint probability distribution over the randvars of a relational data set. Similarly, a unique joint probability distribution for an RDN can be obtained by grounding out the model to obtain a DN and then running an ordered pseudo-Gibbs sampler on the DN. Again, this can be done regardless of whether the model is consistent. The distribution of an inconsistent RDN is the stationary distribution of an ordered pseudo-Gibbs sampler (if it exists) applied to the model.

Learning the structure of an RDN follows the same paradigm as in the propositional case: the CPD for each predicate is learned in turn. Normally, this is done by learning a relational probability tree for each predicate (Neville and Jensen 2007; Natarajan et al. 2012). Section 6 provides a more in-depth discussion of existing RDN structure learning algorithms.

3 Hybrid relational dependency networks

We now describe HRDNs, our proposed extension to RDNs for hybrid domains. First, we describe how to incorporate continuous variables. Second, we describe how to represent the CPDs. Third, we briefly describe how to perform inference in HRDNs.

3.1 Representation

It is relatively natural to extend RDNs to incorporate continuous random variables. It requires modifying the definitions presented in Sect. 2.2.

First, to introduce continuous variables, it suffices to declare the range of a predicate to be an interval of the real numbers. Each continuous randvar associated with such a predicate can then take on any value from this interval. For example, we could define a predicate `numHours/1` with the following random variable declaration:

$$\texttt{random(numHours(C))} \leftarrow \texttt{course(C)}$$

that represents the number of hours needed to study for a course `C`. The range of this predicate can be the following interval:

$$\texttt{range(numHours(C))} = [\texttt{20.0}, \texttt{180.0}]$$

Second, we need to modify the definition of a relational feature to account for the fact that both atoms and aggregation functions can have continuous ranges.

Definition 5 (*Numeric relational feature*) A *numeric relational feature* has the same form, $\mathcal{F}_{L:C,A,\alpha}$, as a discrete relational feature. In contrast to a discrete relational feature, one or both of A and α in a numeric relational feature must have a continuous range.

Example 4 Consider the following numeric relational feature:

$$\mathcal{F}_{\texttt{\{S\}:takes(S,C)=true,numHours(C),average}}$$

This feature computes the average number of hours a student spends studying for all taken classes.

Third, we need to extend the definition of a dependency statement to incorporate numeric relational features.

Definition 6 (*Hybrid dependency statement*) A *hybrid dependency statement* is of the form $\texttt{G} \mid Parents(\texttt{G})$ where G's range may be discrete or continuous and Parents(G) is a set of discrete and/or numeric relational features. Each hybrid dependency statement has an associated CPD.

Note that the type of a CPD for each hybrid dependency is determined according to G's range: for a discrete range it is a probability mass function, and for a continuous range it is a density function.

Now we are ready to formally define an HRDN:

Definition 7 (*HRDN*) An HRDN is a tuple $(\mathcal{P}, RVD, dep)$, where $\mathcal{P}$ is a set of predicates, whose ranges may be discrete or continuous, RVD is a set of randvar declarations and dep is a function mapping each $\texttt{P} \in \mathcal{P}$ to a hybrid dependency statement.

Analogous to an RDN, an HRDN can be viewed as a template for constructing a hybrid dependency network in the following way. The set of predicates $\mathcal{P}$ in an HRDN is split into the set of predicates with discrete range $\mathcal{P}_D$ and the set of predicates with continuous range $\mathcal{P}_C$. Given a set of random variable declarations RVD for all predicates in $\mathcal{P}$ and a set of constants, the set of randvars is $\Phi = \Phi_D \bigcup \Phi_C$ where Φ_D denotes all randvars with discrete ranges and Φ_C denotes all randvars with continuous ranges. The induced hybrid DN will have a node for each randvar in Φ and the parent set of a node is determined in the same manner as described previously for discrete DNs. Each discrete randvar of a predicate $\texttt{P}_d \in \mathcal{P}_\mathcal{D}$ will obtain its own copy of the discrete CPD associated with $\texttt{P}_d$ and each continuous randvar of a predicate $\texttt{P}_c \in \mathcal{P}_\mathcal{C}$ will obtain its own copy of the continuous CPD associated with $\texttt{P}_c$.

A consistent HRDN specifies the joint distribution over the randvars in its corresponding hybrid dependency network. In parallel with the claims of Neville and Jensen (2007), there is a direct correspondence between consistent HRDNs and hybrid Markov logic networks (HMLN) in that the set of distributions that can be encoded by a consistent HRDN is equal to the set of positive distributions that can be encoded with an HMLN with the same adjacencies provided they use the same aggregate functions. If an HRDN induces a hybrid DN that does not contain cycles, then its semantics corresponds to those of a hybrid Bayesian network. Our work primarily considers inconsistent HRDNs. In this case, if there is a stationary distribution of an ordered pseudo-Gibbs sampler applied to an HRDN model, we refer to this distribution as the one represented by the model.

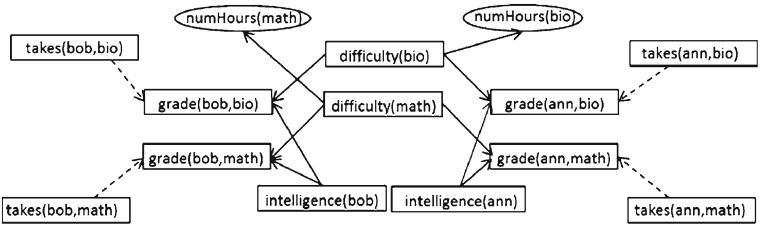

Fig. 2 The ground HRDN specified in Example 5. *Squares* represent randvars with a discrete range, and ovals represent randvars with a continuous range. The *dashed arrows* specify the relevancy condition on `grade/2`

The pseudo-loglikelihood of an HRDN is computed as follows:

$$PLL(M;I) = \sum_{\mathtt{P}_d \in \mathcal{P}_\mathcal{D}} \sum_{g \in gr(\mathtt{P}_d)^I} \log[p(I(\mathtt{g}) \mid I(Parents(\mathtt{g})))] + \sum_{\mathtt{P}_c \in \mathcal{P}_\mathcal{C}} \sum_{g \in gr(\mathtt{P}_c)^I} \log[p(I(\mathtt{g}) \mid I(Parents(\mathtt{g})))]. \quad (7)$$

where the first summation goes over the predicates with a discrete range, and the second goes over the predicates with a continuous range.

Example 5 To illustrate an HRDN, we could extend Example 3 with the `numHours/1` predicate and add the following hybrid dependency statement:

$$\mathtt{numHours(C)} \mid \mathcal{F}_{\{\mathtt{C}\}:\emptyset,\mathtt{difficulty(C)},\mathtt{value}}$$

which states that the number of hours spent studying for a class depends on its difficulty. Figure 2 shows the ground hybrid DN for Example 5. Squares denote randvars with a discrete range and ovals denote randvars with a continuous range.

3.2 Local distributions

Each dependency statement G | Parents(G) has an associated CPD. The type of model used for a CPD depends on both the range of the target atom G and whether Parents(G) contains discrete or numeric features.

In this work, we use a *parametric* approach to density estimation and focus only on variants of Gaussian distributions to model continuous variables. Specifically, we use the following models:

Multinomial If G has a discrete range and its parent set is empty, the CPD is modeled by a multinomial distribution.
Gaussian If G has a continuous range and its parent set is empty, the CPD is modeled by a Gaussian distribution.
Logistic Regression (LR) This CPD is used when the target atom has a discrete range as it facilitates incorporating both discrete and continuous parents (Bishop 1995). Given $range(\mathtt{G}) = \{y_1, y_2, \ldots, y_m\}$, the conditional distribution for the first $(m-1)$ values for a specific grounding $\mathtt{G}\theta$ is:

$$p(\mathtt{G}\theta = y_k \mid Parents(\mathtt{G}\theta)) = \frac{exp\left(w_{k,0} + \sum_{\mathcal{F}\in Parents(\mathtt{G})} w_{k,\mathcal{F}} \cdot \mathcal{F}(\theta)\right)}{1 + \sum_{j=1}^{m-1} exp\left(w_{j,0} + \sum_{\mathcal{F}\in Parents(\mathtt{G})} w_{j,\mathcal{F}} \cdot \mathcal{F}(\theta)\right)}$$

The distribution for the mth value is:

$$p(\mathtt{G}\theta = y_m \mid Parents(\mathtt{G}\theta)) = \frac{1}{1 + \sum_{j=1}^{m-1} exp\left(w_{j,0} + \sum_{\mathcal{F}\in Parents(\mathtt{G})} w_{j,\mathcal{F}} \cdot \mathcal{F}(\theta)\right)} \tag{8}$$

In both equations, $\mathcal{F}$ is a relational feature, $w_{j,\mathcal{F}}$ are the weights associated with $\mathcal{F}$ for value y_j, and $w_{j,0}$ is y_j's bias term.

Linear Gaussian (LG) A linear Gaussian CPD is used when G's range is continuous and all the features in the parent set are numeric (Lauritzen 1992; Koller et al. 1999). An LG is a Gaussian distribution that models μ as a linear combination of the values of the features in the parent set, but assumes a fixed variance σ^2. The distribution is given as:

$$p(\mathtt{G}\theta \mid Parents(\mathtt{G}\theta)) = N\left(w_0 + \sum_{\mathcal{F}\in Parents(\mathtt{G})} w_{\mathcal{F}} \cdot \mathcal{F}(\theta), \sigma^2_{\mathtt{G}}\right) \tag{9}$$

where $\mathcal{F}$ is a numeric feature and $w_{\mathcal{F}}$ is the weight associated with $\mathcal{F}$.

Conditional Linear Gaussian (CLG) A conditional linear Gaussian (CLG) is used if G's range is continuous and its parents set contains a mix of discrete and numeric features. There is a separate linear Gaussian model for every instantiation of the discrete parents. More formally, consider partitioning the parent set of a predicate into the discrete features, $\mathcal{F}_{discrete}$, and the numeric features, $\mathcal{F}_{continuous}$ and let $\mathcal{D}$ be the Cartesian product of ranges of all features in $\mathcal{F}_{discrete}$. Then, the CPD consists of one LG model for each $d \in \mathcal{D}$:

$$p(\mathtt{G}\theta \mid \mathcal{F}_{continuous}, d) = N\left(w_{0_d} + \sum_{F\in\mathcal{F}_{continuous}} w_{\mathcal{F}_d} \cdot \mathcal{F}(\theta), \sigma^2_d\right) \tag{10}$$

Note that because there is a separate LG for each d, each one has an associated variance σ^2_d. A *conditional Gaussian* is a special case of a CLG where the parent set only contains discrete features. Here, a separate Gaussian (mean and variance) is learned for each possible configuration of the parents.

As in the discrete case, it is possible that a feature does not have any groundings. If this occurs and the aggregation function of the feature is not defined on the empty set, then we again return the value *undefined*.

3.3 Inference

Similar to RDNs, inference in HRDNs can be performed by using an ordered pseudo-Gibbs sampler. The difference lies in the fact that HRDNs contain both conditional density functions and probability distributions. Given an HRDN, a set of constants for each type, and possibly a set of relevance conditions, inference is performed as follows.

First, the model is grounded to create the corresponding propositional hybrid dependency network. Second, each randvar gets its own copy of a CPD associated to its predicate. Third, an ordering over the atoms is determined based on the relevance conditions, if specified. This ordering has to ensure that when performing sampling for an atom A we first sample

the values of the atoms in l of the relevance condition $\mathtt{A} \Leftrightarrow l$. For example, consider the relevance condition (4). In each Gibbs sweep, before we sample values for `grade/2` we make sure that the values for `takes/2` are sampled.

Finally, in each Gibbs sweep we visit each ground atom in order and resample its value according to its probability distribution or density function. A randvar is assigned a value from its range or obtains the value *not_relevant* if there exists a relevance condition that is satisfied in the sweep. Each sweep results in an interpretation I and a sample corresponds to only the relevant randvars in I.

4 Structure learning

In this section we present our algorithm for learning the structure of an HRDN. This requires learning a dependency statement and CPD for each predicate in the domain. It is possible to use a decomposable score function to evaluate candidate structures. Thus the problem can be tackled by independently learning a locally optimal CPD for each predicate. Therefore, we refer to our approach as the *Learner of Local Models (LLM)*. When learning the CPD for each predicate, we define a space of candidate features and then greedily select those that improve the score.

Next, we will describe in more detail the key elements of our algorithm, which are (1) its high-level control structure, (2) how to learn a CPD for a single predicate, and (3) how to score the candidate CPDs.

4.1 High-level control structure

Algorithm 1 outlines LLM and it receives as input a set of predicates $\mathcal{P}$, a set of training interpretations D, and a set of validation interpretations V. LLM assumes fully-observed data. At a high level, the algorithm is quite simple. For each predicate $\mathtt{P} \in \mathcal{P}$, it invokes the LearnOneModel function to learn a local distribution that models $\mathtt{P}$ using $\mathcal{P}$. By using a decomposable score function, such as pseudo-loglikelihood, the global score can be optimized by independently finding the best local distribution for each predicate.[2] The final model M is obtained by conjoining all learned local distributions.

Note that this algorithm has the same high-level control structure as existing approaches for learning RDNs. There are two important differences with existing approaches. The first is that the data may contain continuous variables. The second is that, in order to accommodate dependencies on continuous variables, the local distributions are represented via a logistic regression or a (conditional) linear Gaussian as opposed to a relational probability tree.

Next, we describe in detail how to learn and evaluate local distributions.

Algorithm 1: LLM(Predicates $\mathcal{P}$, Training data D, Validation data V)

$M = \{\}$
for all $\mathtt{P} \in \mathcal{P}$ **do**
  CPD_P = LearnOneModel($\mathtt{P}$, $\mathcal{P}$, D, V)
  $M = M \cup \{(P, CPD_{\mathtt{P}})\}$
end for
return: $(\mathcal{P}, M)$

[2] Note that because we use greedy search the learned structure is a local and not a global maximum.

4.2 Learning local distributions

Each learned CPD, regardless of its form, in an HRDN is parameterized by a set of features. Learning the structure of the CPD requires determining which features should appear in the parent set. This can be posed as the problem of searching through the space of candidate features. We adopt a greedy approach that selects one feature at a time to add to the parent set until no inclusion improves the score. Thus, in each iteration, the central procedure is finding the single best feature and adding it to the parent set.

We construct candidate features in the following way. First, let $H = \mathtt{P}(\mathtt{V}_1, \ldots, \mathtt{V}_n)$, where each $\mathtt{V}_i$ is a unique logvar, and let $L = \{\mathtt{V}_1, \ldots, \mathtt{V}_n\}$. Next, we construct all A such that A is different from H. Then, given a user-defined parameter N, for each A all conjunctions of $k \leq N$ randvar-value tests $C = \{(G_1 = v_1), \ldots, (G_k = v_k)\}$ are exhaustively enumerated such that (1) all atoms G_i have a discrete range, (2) no atom G_i is identical to H or A, (3) the set $Q = \{H, G_1, \ldots, G_k, A\}$ is connected.[3] These restrictions ensure that the set of candidate features is finite. For each constructed C and A one candidate feature $\mathcal{F}_{L:C,A,\alpha}$ for each aggregation function α applicable to $range(A)$ is generated. We consider the following aggregation functions:

- If no aggregation is needed, we use $value$,
- If $range(A)$ is discrete and not $\{true, false\}$, we use $mode$,
- If $range(A)$ is discrete and $\{true, false\}$, we use $proportion$ and $exist$,
- If $range(A)$ is continuous, we use $average$, $maximum$, and $minimum$.

The aggregation function $proportion$ computes the proportion of a feature's possible groundings that are true. The other functions take on their traditional meanings.

Algorithm 2 outlines our procedure for learning the dependency for a predicate $\mathtt{P}$. As input, it receives the target predicate $\mathtt{P}$, the full set of predicates $\mathcal{P}$ for the domain, a training set D, and a validation set V. First, the algorithm starts by constructing the set of candidate features for $\mathtt{P}$. Second, it repeatedly iterates through the set of candidate features and evaluates the utility of adding each feature to the parent set. Each feature addition is followed by learning the CPD on the training data D and then scoring it on the validation data V. In each iteration, the single best feature is added to the parent set. If no feature improves the score, the procedure terminates. Note that the form of the CPD depends on both $\mathtt{P}$ and the features in the parent set. If $\mathtt{P}$'s range is discrete, then the CPD is represented via logistic regression. If $\mathtt{P}$'s range is continuous, we use linear Gaussians if the parents only contain numeric features and conditional linear Gaussians when the parent set contains both numeric and discrete features.

The two following subsections explain how we estimate the parameters of the CPDs using the training data and how we evaluate the local models.

4.3 Estimating the parameters for candidate CPDs

Next, we briefly describe how to estimate the parameters for the CPDs for the different types of dependency statements that may appear in a learned HRDN.

Multinomial The maximum likelihood parameters of the multinomial are learned from the data.

Gaussian The maximum likelihood estimates of the Gaussian's mean and the variance are learned from the data.

[3] Here, we mean connected in the sense that the graph (Q, E) is connected with $E = \{\{u, v\} \mid u, v \in Q \wedge u \text{ and } v \text{ share variables}\}$.

Algorithm 2: LearnOneModel(Target Predicate P, All Predicates $\mathcal{P}$, Training Data D, Validation Data V)

$Parents(\mathtt{P}) = \emptyset$
$CPD_{\mathtt{P}} =$ learnCPD($Parents(\mathtt{P})$, D)
$FS =$ GenerateCandidateFeatures(P, $\mathcal{P}$)
repeat
  $F_{best} = null$
  $CPD_{best} = CPD_{\mathtt{P}}$
  for F in FS **do**
    $CPD_{temp} =$learnCPD($Parents(\mathtt{P}) \cup \{F\}$, D)
    if $score(CPD_{temp}, V) > score(CPD_{\mathtt{best}}, V)$ **then**
      $CPD_{best} = CPD_{temp}$
      $F_{best} = F$
    end if
  end for
  if $F_{best} \neq null$ **then**
    $Parents(\mathtt{P}) = Parents(\mathtt{P}) \cup \{F_{best}\}$
    $CPD_{\mathtt{P}} = CPD_{best}$
    $FS = FS \setminus \{F_{best}\}$
  end if
until $F_{best} = null$
return: $CPD_{\mathtt{P}}$

Logistic regression Parameter estimation requires learning the weight vectors for the logistic regression model. We follow the standard approach and take the (partial) derivative of the conditional loglikelihood of the data and perform gradient ascent to estimate the weights (Mitchell 1997).
Linear Gaussian Parameter learning requires estimating the weight vector for the linear regression model. This can be done via standard techniques for training a linear regressor and we use ridge regression (Bishop 1995). We estimate the variance by computing the expected value of the squared difference between the actual value and the model's predicted value.
Conditional linear Gaussian In CLGs, each configuration of the discrete parents has an associated LG model. The parameters for each LG model are learned as described above.

4.4 Evaluating candidate models

Traditionally, a candidate model is evaluated using a score function that trades off the model's fit to the data versus some penalty term based on the model's complexity to avoid overfitting. For a candidate model M, we use the following score function, which is based on the Minimum Description Length (MDL) (Schwarz 1978):

$$MDL(M, D) = PLL(M, D) - Penalty(M, D) \tag{11}$$

where $PLL(M, D)$ is computed using Eq. (6) and $Penalty(M, D)$ is the following penalty term:

$$Penalty(M, D) = \frac{1}{2} \sum_{I \in D} \sum_{\mathtt{P} \in \mathcal{P}} log_2(|gr(\mathtt{P})|^I) \cdot B_{\mathtt{P}} \cdot K$$

where $|gr(\mathtt{P})|^I$ is the number of relevant randvars of predicate P in interpretation I, $B_\mathtt{P}$ is the number of free parameters in P's CPD and K is the size of P's CPD.[4] Next, we will explain in more detail how $B_\mathtt{P}$ and K are calculated.

When the CPD for P is represented by a logistic regression model (see Eq. 8), the number of free parameters is:

$$B_\mathtt{P} = (|range(\mathtt{P})| - 1) \cdot (1 + |Parents(\mathtt{P})|)$$

where $(1 + |Parents(\mathtt{P})|)$ is the number of weights that must be learned to parameterize the model (i.e., one for each feature plus the intercept). For continuous CPDs, this is slightly more involved to compute. For an LG, the number of free parameters is:

$$B_\mathtt{P} = 1 + (1 + |Parents(\mathtt{P})|)$$

where the first 1 is for the variance σ^2 and $(1 + |Parents(\mathtt{P})|)$ is the number of weights that must be learned to parameterize the model (i.e., one for each feature in the parent set plus the intercept). Recall that in a CLG, one LG model is learned for each possible instantiation of the discrete parents. Thus the number of free parameters for a CLG is:

$$B_\mathtt{P} = d \cdot (1 + (1 + |Parents_C(\mathtt{P})|))$$

where d is the number of elements in the Cartesian product of the ranges of the discrete parents, $Parents_C(\mathtt{P})$ denotes only numeric features in the parent set of P and $(1 + (1 + |Parents_C(\mathtt{P})|))$ is the number of parameters needed to model each LG.

The size K of P's CPD is the sum of the feature lengths in the parent set:

$$K = \sum_{\mathcal{F} \in Parents(\mathtt{P})} |\mathcal{F}_{L:C,A,\alpha}| \tag{12}$$

where $|\mathcal{F}_{L:C,A,\alpha}| = |C| + 1$ is the length of a feature.

5 Experiments

This section empirically evaluates our HRDN structure learning algorithm LLM. Specifically, we want to answer the following questions:

1. How does varying the amount of training data affect the quality of the learned model and the run time of the learning algorithm?
2. Do we learn more accurate models by learning a hybrid model (i.e., explicitly modeling continuous variables) or by discretizing all continuous variables prior to learning?
3. How does our approach compare to MLN (Richardson and Domingos 2006) structure learning?

All our code, data and models are publicly available.[5] We first describe the data sets we will use and then explain the experimental setup. Finally, we present and discuss the results.

5.1 Data sets

We use one synthetic and one real-world data set to answer these questions.

[4] Because we assume that all variables are observed, we do not need to run Gibbs sampling to compute the PLL.

[5] http://dtai.cs.kuleuven.be/ml/systems/llm.

Table 1 Data set characteristics for the synthetic data when varying the number of interpretations used for learning

#Interpretations	Average Sum of #*Randvars*
1	19,627
2	39,308
4	79,027
8	157,959
16	315,334

#Interpretations is the number of training interpretations. Average Sum #*Randvars* is the number of randvars summed across all training interpretations averaged over the ten generated data sets. Each interpretation has 100 students, 50 courses and 50 professors objects

Table 2 Data set characteristics for the synthetic data when varying the domain size of each object type in the training interpretation

#*Students*	#*Courses*	#*Professors*	Average #*Randvars*
100	50	50	19,548
200	75	75	66,577
400	100	100	226,679
800	125	125	796,328

#*Students*, #*Courses*, and #*Professors* report the number of each type of object. Average #*Randvars* is the number of randvars averaged over the ten data sets generated for each domain size

Synthetic university data We used a modified version of the well-known university model (Getoor et al. 2001) to generate synthetic data. We made the following alterations. First, we switched the range of `intelligence/1` from discrete to continuous. Second, we added two predicates with continuous ranges: `numHours/1`, which is the estimated number of hours a student needs to study for a course, and `ability/1`, which is the ability of a professor. Finally, we added a Boolean predicate `friend/2`, which denotes whether two students are friends. Appendix 1 contains a complete description of the model.

We generate synthetic data in two ways. First, we fix the domain size of each type within an interpretation and vary the number of training interpretations. We learn models by using one, two, four, eight and 16 interpretations. We use one validation and one test interpretation. Second, we fix the number of training and validation interpretations to one and vary the domain size of each object. The learned models in this setup are evaluated on a test interpretation consisting of 800 students, 125 courses and 125 professors. Tables 1 and 2 show the characteristics of the domains for the first and second synthetic setup, respectively.

For each experimental condition, we repeat the following process ten times. We generate the appropriate number of interpretations, where each interpretation is constructed by performing 2000 iterations of the ordered pseudo-Gibbs sampling (see Sect. 3.3) using the handcrafted model and the specified number of constants.

For each generated data set, we also create a corresponding discretized version by binning each continuous randvar into a number of equal-size intervals. We used 2, 4, 6 and 8 bins.

Real-world PKDD'99 financial data set Our real-world domain is the financial data set from the PKDD'99 Discovery Challenge (Berka 1999). It consists of services one bank offers its clients such as loans, accounts, and credit cards among others. In the original data, the

Table 3 Characteristics of the PKDD'99 financial data set

#*Account*	#*Loan*	#*Client*	#*District*	#*Randvars*
4,490	680	5,358	77	3,157,657

#*Account*, #*Loan*, #*Client*, and #*District* report the number of objects of each type and #*Randvars* is the number of randvars in the data set

transaction table contains more than one million transactions. Therefore, we introduced several predicates (e.g., average of monthly withdrawals for an account) to summarize the information contained in this table. This results in 16 predicates[6] about four types of objects: *clients*, *accounts*, *loans* and *districts*. Ten predicates have a continuous range and six have a discrete range.

We consider *account* to be the central object type in the PKDD'99 financial data set. The original data set consists of 4500 accounts, but we omit ten accounts that have missing data. We then split the data associated with these accounts into tenfolds. To avoid leakage of information, all information about clients, loans and districts related to one account appear in the same fold. We used sixfolds for training, threefolds for validation and one for testing. Table 3 reports the characteristics of this data set.

Again, we create a discretized version of the data by binning each continuous randvar into a number of equal-size intervals and used 2, 4, 6 and 8 bins.

5.2 Methodology

We compare the following four learners on all experiments:

LLM-H This corresponds to learning a model using our LLM algorithm on the data containing both continuous and discrete variables.
LLM-D This corresponds to learning a model using our LLM algorithm on the discretized data. Thus each learned local distribution is modeled using a logistic regression CPD.
LSM This corresponds to learning a model using the publicly available implementation of LSM (Kok and Domingos 2010) on the discretized data. LSM is the state-of-the-art Markov logic network structure learning algorithm.
Independent This learner constructs a model on the hybrid data such that all randvars are independent. That is, it models the joint distribution as a product of marginal distributions.

On the experiments involving the PKDD'99 financial data set, we include an additional baseline: a handcrafted model. We built a local model to predict each predicate by a set of handcrafted non-relational features. These features are used to predict a property of an object by means of some other properties of that object. The features can be found in Appendix 4. For predicates with a discrete range, we used logistic regression. For predicates with a continuous range, we used both linear regression and MP5 (a regression tree) as implemented in Weka (Hall et al. 2009).

Experimental details LLM is implemented as a combination of Java and Prolog. Java is used for performing the learning and Prolog is used to compute the value of a feature. When generating features, we set the length of the features to be at most $N = 3$. Usually, in relational domains, only a small fraction of the Boolean atoms is true (e.g., the number of people who are friends is quite sparse compared to the number of possible friendships). Therefore, for

[6] Table 11 in "Appendix 2" describes the predicates.

efficiency reasons, we subsample the false Boolean atoms during learning (Natarajan et al. 2012) to achieve a 1:1 ratio of true to false groundings in all experiments.

For LSM, we contacted the authors in order to know what the most important parameters were to tune. Then, we tried several parameter combinations, and used the validation data to select appropriate ones for each data set.

Evaluation metrics We evaluate the quality of the learned models using several metrics. First, to measure the quality of the probability estimates, we report the weighted pseudo-loglikelihood (WPLL) (Kok and Domingos 2005). This corresponds to calculating the PLL of an interpretation as the sum of PLLs for each predicate divided by the number of groundings of that predicate in the interpretation.

Second, to measure the predictive performance, we report the area under the ROC curve (AUC-ROC) for discrete predicates and the normalized root-mean-square error (NRMSE) for continuous predicates. Because we have multi-class categorical variables in our domains, we calculate the multi-class AUC-ROC (Domingos and Provost 2000), which we denote as AUC_{total}. The NRMSE for a predicate ranges from zero to one and is calculated by dividing RMSE by the predicate's range.

Additionally, since we know the model structure for the synthetic data, we compare how closely the learned model reflects the handcrafted structure using the following edit distance. For each predicate, we compare the true parent set to the learned parent set. For each feature in the true parent set, we find its closest feature in the learned parent set according to the following distance metric. The distance Δ between two features, $\mathcal{F}_{L_1:C_1,A_1,\alpha_1}$ and $\mathcal{F}_{L_2:C_2,A_2,\alpha_2}$, is calculated as:

$$\Delta(\mathcal{F}_1, \mathcal{F}_2) = |C_1 \backslash C_2| + |C_2 \backslash C_1| + \delta_{A_1,A_2} + \delta_{\alpha_1,\alpha_2}$$

where δ_{A_1,A_2} equals zero if the two atoms A_1 and A_2 originate from the same predicate and their logvars are equivalent, otherwise it equals one. Similarly, $\delta_{\alpha_1,\alpha_2}$ equals zero if α_1 and α_2 represent the same aggregation function, otherwise it equals one. When the best match is found, both the true and the learned feature are excluded from further comparisons, and the edit distance is incremented by the distance between them. Furthermore, the final distance is incremented by the length of each feature that must be added or removed from the learned dependency parent set.

We use a one-tailed paired t test to assess the significance of the results obtained through ten independent runs for the synthetic experimental setup and tenfolds for the real-world data set. The null hypothesis states that there is no difference between two approaches and we reject it when $p<0.01$. For all metrics, we report the metric itself along with its standard deviation.

5.3 Results and discussion

We now present experimental results for the synthetic and real-world data sets.

Results on synthetic data Table 4 shows how the WPLL of each approach varies as a function of the number of training interpretations. Learning from the hybrid data results in a significantly more accurate learned model than learning from the discretized data in all cases except for one in which we have one training interpretation and six discretizing bins. When using the same number of bins for discretization, LLM-D learns more accurate models than LSM on all settings. Note that LSM ran out of memory on all runs when training on eight and 16 interpretations. Finally, all learning approaches always outperform the no-learning baseline.

Table 4 The WPLL on the synthetic data as a function of the number of training interpretations

	Nr. training interpretations				
	1	2	4	8	16
LLM-H	**−18.22 ± 0.5**	**−18.16 ± 0.5**	**−17.89 ± 0.2**	**−17.87 ± 0.2**	**−17.83 ± 0.3**
LLM-D(#bins=2)	−21.33 ± 0.3*	−21.10 ± 0.3*	−21.06 ± 0.3*	−21.06 ± 0.3*	−21.05 ± 0.2*
LLM-D(#bins=4)	−19.53 ± 0.6*	−19.27 ± 0.3*	−19.20 ± 0.3*	−19.12 ± 0.3*	−19.04 ± 0.3*
LLM-D(#bins=6)	−19.34 ± 0.9	−18.77 ± 0.4*	−18.56 ± 0.3*	−18.55 ± 0.3*	−18.52 ± 0.3*
LLM-D(#bins=8)	−20.03 ± 1.0*	−19.11 ± 0.8*	−18.64 ± 0.5*	−18.62 ± 0.5*	−18.30 ± 0.3*
LSM(#bins=2)	−23.33 ± 0.2*†	−23.28 ± 0.2*†	−22.86 ± 0.8*†	OoM	OoM
LSM(#bins=4)	−23.00 ± 0.3*†	−22.86 ± 0.4*†	−21.65 ± 1.3*†	OoM	OoM
LSM(#bins=6)	−22.79 ± 0.4*†	−22.67 ± 0.4*†	−22.00 ± 1.3*†	OoM	OoM
LSM(#bins=8)	−22.62 ± 0.3*†	−22.62 ± 0.9*†	−21.27 ± 1.4*†	OoM	OoM
Independent	−23.68 ± 0.4*	−23.63 ± 0.4*	−23.53 ± 0.4*	−23.54 ± 0.4*	−23.52 ± 0.4*

The best WPLLs are in bold, an asterisk (*) denotes significantly worse results for $p<0.01$ compared to LLM-H. A dagger (†) denotes when LSM performs significantly worse than LLM-D for $p<0.01$ on the data discretized with the same number of bins. OoM denotes out of memory

Table 5 The run times in minutes on the synthetic data as a function of the number of training interpretations

	Nr. training interpretations				
	1	2	4	8	16
LLM-H	15.58 ± 1.8	18.72 ± 2.3	28.53 ± 2.7	**48.16 ± 3.2**	**76.51 ± 3.51**
LLM-D(#bins=2)	21.82 ± 4.3	30.48 ± 5.5	53.03 ± 13.7	85.74 ± 10.5	140.62 ± 19.2
LLM-D(#bins=4)	28.71 ± 5.7	40.24 ± 7.3	70.54 ± 17.5	109.80 ± 21.0	162.13 ± 41.0
LLM-D(#bins=6)	35.69 ± 7.1	49.84 ± 9.6	83.34 ± 24.0	135.73 ± 13.1	201.40 ± 139.2
LLM-D(#bins=8)	42.70 ± 8.5	57.63 ± 14.4	98.33 ± 29.1	166.11 ± 38.4	255.90 ± 46.7
LSM(#bins=2)	3.73 ± 0.1	6.70 ± 0.0	**7.48** ± 0.1	OoM	OoM
LSM(#bins=4)	3.44 ± 0.1	**6.22** ± 0.1	8.49 ± 0.0	OoM	OoM
LSM(#bins=6)	**3.22 ± 0.0**	6.23 ± 0.0	12.65 ± 0.0	OoM	OoM
LSM(#bins=8)	4.34 ± 0.1	6.27 ± 0.1	13.33 ± 0.1	OoM	OoM

The best run times are in bold and OoM denotes out of memory

Table 5 presents the run times for all algorithms as a function of increasing the number of training interpretations. LSM is the fastest learner, but it produces lower-quality models. For all approaches, the run time scales linearly with the number of interpretations. Learning an HRDN is always faster than learning an RDN. When discretizing the data, the run time is influenced by the number of bins used: the more bins there are, the slower the discrete learner is. This occurs because adding more bins increases the size of the search space.

Finally, Fig. 3 shows how the edit distance varies as a function of the number of training interpretations. As expected, the edit distance decreases as more training data are used.

Table 6 shows the WPLLs of all learners as a function of increasing the domain size for each object. To encapsulate the effect of domain size changes in a single number, we use the number of randvars in an interpretation. Again, we see that all the learners outperform the

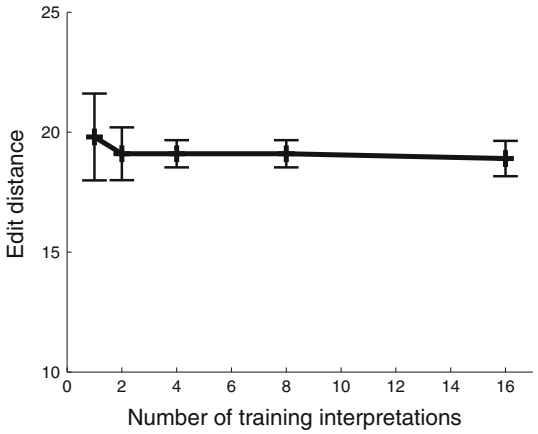

Fig. 3 The effect of the number of training interpretations on the average edit distance between the handcrafted HRDN model and the hybrid model learned with LLM-H

Table 6 The WPLL on the synthetic data as a function of the domain size

	Domain size (#students × #courses × #professors)			
	100×50×450	200×75×75	400×100×100	800×125×125
LLM-H	**−18.11 ± 0.3**	**−17.84 ± 0.2**	**−17.78 ± 0.2**	**−17.72 ± 0.3**
LLM-D(#bins=2)	−20.56 ± 0.5*	−20.47 ± 0.2*	−20.42 ± 0.2*	−20.54 ± 0.2*
LLM-D(#bins=4)	−18.95 ± 0.8*	−18.50 ± 0.2*	−18.48 ± 0.2*	−18.60 ± 0.3*
LLM-D(#bins=6)	−18.62 ± 0.9*	−18.22 ± 0.6	−17.86 ± 0.2	−17.87 ± 0.2
LLM-D(#bins=8)	−19.39 ± 0.8*	−18.17 ± 0.6	−18.05 ± 0.6	−17.86 ± 0.4
LSM(#bins=2)	−24.45 ± 0.2*†	−22.58 ± 0.1*†	−22.53 ± 0.2*†	−22.72 ± 0.2*†
LSM(#bins=4)	−23.00 ± 0.3*†	−23.15 ± 0.5*†	−22.20 ± 0.2*†	−21.83 ± 0.2*†
LSM(#bins=6)	−22.79 ± 0.4*†	−25.55 ± 0.3*†	−21.83 ± 0.2*†	−21.92 ± 0.2*†
LSM(#bins=8)	−22.62 ± 0.3*†	−25.64 ± 0.1*†	−21.71 ± 0.2*†	−21.79 ± 0.2*†
Independent	−23.55 ± 0.1*	−23.48 ± 0.1*	−23.46 ± 0.1*	−23.42 ± 0.1*

The best WPLLs are in bold, an asterisk (*) denotes significantly worse results for $p < 0.01$ compared to LLM-H. A dagger (†) denotes when LSM performs significantly worse than LLM-D for $p < 0.01$ on the data discretized with the same number of bins

independent model. LLM-H always learns significantly more accurate models than LSM. LLM-H learns a significantly more accurate model than LLM-D except when discretizing the data into 6 or 8 bins on the data sets with 200, 400 and 800 students.

Table 7 shows the run time of all approaches as a function of increasing domain size. Similar to the previous setup, LSM exhibits better run times than either LLM-H or LLM-D, but it produces lower-quality models. As expected, both LLM-H and LLM-D run time varies quadratically with the increase in domain size. LSM's run time seems to vary linearly, which probably occurs due to its random-walk style search for patterns, which does not necessarily examine all the variables in the training database. When learning (H)RDNs, LLM-H is faster than LLM-D. Again, in general, increasing the number of bins increases the training time.

Figure 4 shows that the edit distance between LLM-H's learned model and the handcrafted model decreases as the number of randvars in the training interpretation increases. More

Table 7 The run times in minutes on the synthetic data as a function of the domain size for all the learners

	Domain size (#students × #courses × #professors)			
	100×50×50	200×75×75	400×100×100	800×125×125
LLM-H	15.58 ± 1.8	54.61 ± 3.2	171.82 ± 21.5	2171.05 ± 306.3
LLM-D(#bins=2)	21.82 ± 4.3	98.68 ± 5.0	270.81 ± 38.7	2164.11 ± 154.6
LLM-D(#bins=4)	28.71 ± 5.7	128.51 ± 8.6	341.76 ± 40.7	3026.22 ± 317.4
LLM-D(#bins=6)	35.69 ± 7.1	156.98 ± 8.1	476.16 ± 50.8	2923.45 ± 150.7
LLM-D(#bins=8)	42.71 ± 8.5	182.18 ± 5.8	700.07 ± 80.8	4119.31 ± 387.3
LSM(#bins=2)	3.73 ± 0.1	6.13 ± 0.1	11.85 ± 0.1	**18.34 ± 0.3**
LSM(#bins=4)	3.44 ± 0.1	**5.18 ± 0.1**	**10.32 ± 0.1**	19.72 ± 0.2
LSM(#bins=6)	**3.22 ± 0.0**	5.74 ± 0.1	11.80 ± 0.1	19.68 ± 0.2
LSM(#bins=8)	4.34 ± 0.0	5.71 ± 0.1	11.33 ± 0.1	20.68 ± 0.2

The best run times are in bold

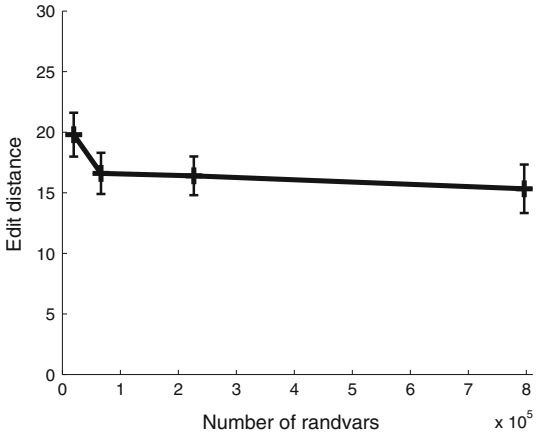

Fig. 4 The effect of increasing the domain size of each object type on the average edit distance between the handcrafted HRDN model and the hybrid model learned with LLM-H. We summarize the effect of changing the domain sizes by showing the number of randvars in the training interpretation

(observed) random variables equates to more training data, and, as expected, more data allows us to learn more accurate models.

In both synthetic setups, we noticed that in the learned model `difficulty(C)` depends on `nrhours(C)`. This dependency is not encoded explicitly in the handcrafted model. However, `nrhours(C)` does depend on `difficulty(C)` in the original model. In both cases, this contributes to the edit distance.

More detailed results for both synthetic setups can be found in Appendix 3.

Results on the PKDD'99 financial data set Figure 5 shows the WPLL for all approaches on the PKDD'99 financial data set as a function of the number of bins used for discretization. For the handcrafted models, we denote the combination of logistic regression and linear regression as LR+LinR, and the combination of logistic regression and MP5 regression trees with LR+MP5. In the figure, the lines for LLM-H, LR+LinR, LR+MP5 and the independent model are straight because these approaches operate directly on the hybrid data and hence do not perform discretization. We see a clear ranking between the approaches: LLM-H > LR+LinR > LR+MP5 > LLM-D > LSM > independent.

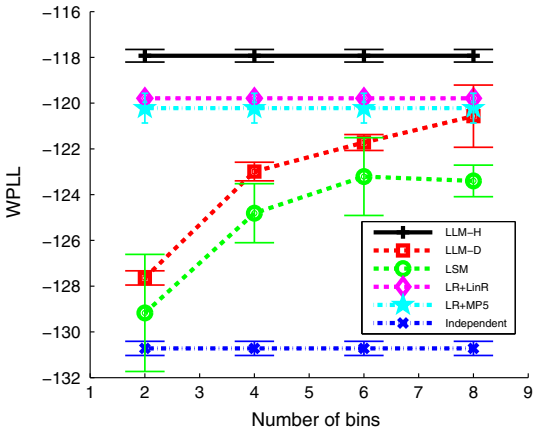

Fig. 5 The WPLL for each approach on the PKDD'99 financial data set as a function of the number of bins used for discretization. Note that the results for LLM-H, LR+LinR, LR+MP5 and the independent model do not depend on the number of bins used for discretization

Table 8 The performance of the two variants of the handcrafted models, LR+LinR and LR+MP5, compared to LLM-H on the hybrid data for the PKDD'99 financial data set

Evaluation	Predicate	LR+LinR	LR+MP5	LLM-H
AUC_{total}	`clientDistrict/2`	$0.59 \pm 0.02^*$	$0.59 \pm 0.02^*$	$\mathbf{0.64 \pm 0.02}$
	`gender/1`	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.01}$
	`hasAccount/2`	$0.50 \pm 0.01^*$	$0.50 \pm 0.01^*$	$\mathbf{0.56 \pm 0.01}$
	`freq/1`	$\mathbf{0.86 \pm 0.01}$	$\mathbf{0.86 \pm 0.01}$	$0.82 \pm 0.01^*$
	`hasLoan/2`	$0.76 \pm 0.01^*$	$0.76 \pm 0.01^*$	$\mathbf{1.00 \pm 0.01}$
	`loanStatus/1`	$\mathbf{0.79 \pm 0.03}$	$\mathbf{0.79 \pm 0.03}$	$0.66 \pm 0.04^*$
NRMSE	`clientAge/2`	0.28 ± 0.03	$\mathbf{0.28 \pm 0.01}$	0.28 ± 0.02
	`avgSalary/1`	$0.13 \pm 0.01^*$	$\mathbf{0.11 \pm 0.01}$	$0.13 \pm 0.02^*$
	`ratUrbInhab/1`	$0.20 \pm 0.01^*$	$\mathbf{0.15 \pm 0.00}$	$0.20 \pm 0.00^*$
	`avgSumOfW/1`	$\mathbf{0.02 \pm 0.00}$	$0.03 \pm 0.00^*$	$\mathbf{0.02 \pm 0.00}$
	`avgSumOfCred/1`	0.02 ± 0.01	$0.03 \pm 0.00^*$	$\mathbf{0.02 \pm 0.00}$
	`stdOfW/1`	0.05 ± 0.01	$\mathbf{0.05 \pm 0.00}$	0.05 ± 0.01
	`stdOfCred/1`	0.05 ± 0.01	$\mathbf{0.04 \pm 0.01}$	0.05 ± 0.01
	`avgNrWith/1`	$0.12 \pm 0.02^*$	$\mathbf{0.10 \pm 0.00}$	$0.15 \pm 0.01^*$
	`loanAmount/1`	0.15 ± 0.02	$\mathbf{0.15 \pm 0.01}$	0.16 ± 0.02
	`monthlyPayments/1`	0.17 ± 0.02	$\mathbf{0.17 \pm 0.01}$	0.18 ± 0.02

LR+LinR uses logistic regression for discrete predicates and linear regression for continuous predicates, and LR+MP5 uses logistic regression for discrete predicates and regression trees for continuous predicates. The best results are in bold and an asterisk (*) denotes the result that is significantly worse ($p < 0.01$) than the best result

Table 8 shows the (multi-class) AUCs and NRMSE for LLM-H and the handcrafted models. All three approaches tend to have similar results on most predicates. Note that the handcrafted features used to propositionalize the data are all features that LLM-H is able to learn automatically.

Table 9 reports the AUC_{total} for LLM-H, LLM-D and LSM. Out of the six discrete predicates, LLM-H has a higher AUC_{total} on one predicate, the same on two and worse on

Table 9 AUC_{total} results for LLM-H, LLM-D and LSM on the six discrete predicates in the PKDD'99 financial data set

		Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	LLM-H	LLM-D	LSM	LLM-D	LSM	LLM-D	LSM	LLM-D	LSM
clientAge/2	–	0.49 ± 0.04	0.50 ± 0.00	$\mathbf{0.51 \pm 0.04}$	0.50 ± 0.00	0.49 ± 0.03	0.50 ± 0.00	0.50 ± 0.01	0.50 ± 0.00
clientDistrict/2	$\mathbf{0.64 \pm 0.02}$	0.56 ± 0.01	0.50 ± 0.00	0.56 ± 0.01	0.50 ± 0.00	0.55 ± 0.01	0.50 ± 0.00	0.56 ± 0.01	0.50 ± 0.00
gender/1	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$
hasAccount/2	$\mathbf{0.56 \pm 0.01}$	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00
avgSalary/1	–	0.87 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.67 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.59 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.56 ± 0.00	0.49 ± 0.00
ratUrbInhab/1	–	$\mathbf{0.60 \pm 0.00}$	$\mathbf{0.60 \pm 0.00}$	0.52 ± 0.00	0.50 ± 0.00	0.51 ± 0.00	0.50 ± 0.00	0.51 ± 0.00	0.50 ± 0.00
avgSumOfW/1	–	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	0.85 ± 0.02	$\mathbf{0.99 \pm 0.00}$	0.64 ± 0.00	$\mathbf{0.99 \pm 0.00}$	0.63 ± 0.02	$\mathbf{0.99 \pm 0.01}$
avgSumOfCred/1	–	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	0.96 ± 0.01	$\mathbf{0.99 \pm 0.00}$	0.80 ± 0.06	$\mathbf{0.99 \pm 0.00}$	0.66 ± 0.09	$\mathbf{0.99 \pm 0.00}$
stdOfW/1	–	0.90 ± 0.02	$\mathbf{0.98 \pm 0.01}$	0.88 ± 0.06	0.97 ± 0.01	0.65 ± 0.05	0.97 ± 0.01	0.60 ± 0.01	0.95 ± 0.01
stdOfCred/1	–	0.91 ± 0.04	0.96 ± 0.06	0.81 ± 0.04	0.96 ± 0.03	0.68 ± 0.01	$\mathbf{0.98 \pm 0.01}$	0.64 ± 0.04	0.97 ± 0.01
freq/1	0.82 ± 0.01	0.64 ± 0.03	0.86 ± 0.17	0.59 ± 0.05	0.78 ± 0.15	0.57 ± 0.03	$\mathbf{0.88 \pm 0.10}$	0.56 ± 0.03	0.86 ± 0.10
avgNrWith/1	–	0.57 ± 0.03	0.58 ± 0.36	0.56 ± 0.03	$\mathbf{0.67 \pm 0.20}$	0.49 ± 0.02	0.59 ± 0.11	0.51 ± 0.01	0.59 ± 0.18
hasLoan/2	$\mathbf{1.00 \pm 0.01}$	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00
loanAmount/1	–	$\mathbf{0.86 \pm 0.05}$	0.78 ± 0.31	0.72 ± 0.02	0.65 ± 0.25	0.53 ± 0.04	0.67 ± 0.19	0.55 ± 0.03	0.50 ± 0.00
loanStatus/1	0.66 ± 0.04	0.56 ± 0.05	0.70 ± 0.25	0.59 ± 0.06	0.75 ± 0.33	0.62 ± 0.05	0.72 ± 0.33	0.51 ± 0.07	$\mathbf{0.78 \pm 0.34}$
monthlyPayments/1	–	0.67 ± 0.06	$\mathbf{0.77 \pm 0.22}$	0.66 ± 0.03	0.68 ± 0.19	0.66 ± 0.03	0.52 ± 0.06	0.63 ± 0.03	0.50 ± 0.00

The best results are in bold

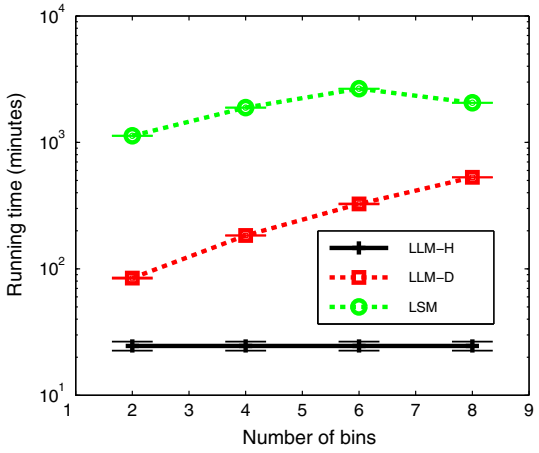

Fig. 6 The run time of each approach on the PKDD'99 financial data set as a function of the number of bins used for discretization. The y-axis (run time) is on a log scale. Note that LLM-H's results do not depend on the number of bins used for discretization

three compared to LLM-D. Compared to LSM, it wins on three predicates, loses on two and draws on one.

Figure 6 shows the run times for this data set as a function of the number of bins used for discretization. LLM-H exhibits better run times than both LLM-D and LSM. LSM is faster than LLM-D except when discretizing the data into two bins.

When we inspected the models learned on the PKDD'99 financial data set, we found a considerable number of bi-directional dependencies. This means that our algorithm succeeded in learning a model that is mostly structurally consistent. For example, it learned that the monthly payment amount for a loan depends on the loan amount, and vice versa. The same holds for the average salary and the ratio of urban inhabitants in a district, the average amount withdrawn from an account and the average amount credited to an account, the average amount withdrawn from an account and the average number of withdrawals for an account, among others.

More detailed results for the PKDD'99 financial data set can be found in Appendix 3.

Discussion Now we can revisit and answer the three experimental questions posed at the beginning of this section. To address the first question, we used the synthetic data to explore the scaling behavior of our algorithm. We found that as the amount of training data increases both the accuracy of the learned models and their faithfulness to the ground truth model slightly improve.

The second question revolves around whether it is better to learn from hybrid data or discretized data. On all experiments, we have seen that learning from the hybrid data directly consistently results in significantly more accurate learned models (according to WPLL) than discretizing the data prior to learning. Finally, we wanted to compare our proposed learning algorithm to the state-of-the-art MLN learner. The results show that on both hybrid and discrete data LLM learns more accurate models than LSM.

6 Related work

On the propositional level, researchers have considered extending formalisms such as Bayesian networks and dependency networks to model both discrete and continuous

distributions. In terms of hybrid Bayesian networks, most of the work has focused on inference (Koller et al. 1999; Yuan and Druzdzel 2007; Murphy 1998; Moral et al. 2001; Lauritzen and Jensen 2001). There have also been some initial attempts for parameter learning (Murphy 1998) and structure learning (Romero et al. 2006). Cobb et al. (2007) provides a more detailed overview of work on hybrid Bayesian networks.

There has been some work on structure learning for hybrid dependency networks. Dobra (2009) has proposed bounded stohastic search for variable selection (structure learning) for sparse genetic dependency networks that contain both discrete and continuous variables. Meinshausen and Bühlmann (2006) use neighbourhood selection with the Lasso for structure learning as a computationally attractive alternative to standard covariance selection methods for multivariate normal distributions. Guo and Gu (2011) use dependency networks for multi-label classification where each CPD represents a probabilistic or non-probabilistic binary classifier that can have both discrete and continuous predictors.

Our work represents a relational approach and builds off of two lines of research: structure learning for RDNs and hybrid relational probabilistic models. There are two existing structure learning approaches for RDNs (Neville and Jensen 2007; Natarajan et al. 2012). Both approaches perform structure learning by finding the best conditional distribution independently for each predicate. They slightly differ in how they represent the CPDs. Neville and Jensen (2007) learn a single relational probability tree (Neville et al. 2003) for each predicate. Natarajan et al. (2012) represent individual conditional distributions as a weighted sum of relational regression trees (Blockeel and De Raedt 1998), which are learned by a stage-wise optimization procedure. However, these approaches do not explicitly model continuous distributions and instead require them to be discretized. In contrast, our approach is able to directly encode dependencies between discrete and continuous random variables without discretization. Doing so necessitates representing the CPDs with logistic regression or conditional (linear) Gaussian model as opposed to a relational probability tree.

There are several formalisms that can represent hybrid relational domains including Hybrid Markov Logic Networks (HMLNs) (Wang and Domingos 2008), Hybrid Problog (HProblog) (Gutmann et al. 2011), Continuous Bayesian Logic Programs (CBLPs) (Kersting and De Raedt 2001), Learning Modulo Theories (LMT) (Teso et al. 2013) and Hybrid Probabilistic Relational Models (HPRMs) (Narman et al. 2010). Additionally, formalisms such as Relational Continuous Models (RCMs) (Choi et al. 2010) and Gaussian Logic (Kuželka et al. 2011) can model domains that exclusively contain continuous variables. The latter formalism also provides support for structure learning. Most of these formalisms focus on representation and reasoning issues in hybrid relational domains. HMLNs, CBLPs and LMTs also provide support for learning the parameters of a given model from data. Next, we provide a more detailed comparison between our approach and HMLNs, HProblog and CBLPs.

Representationally, HMLNs, CBLPs and HRDNs all serve as template languages for constructing a different type of propositional graphical model. Hence, each formalism inherits the strengths and weaknesses of the underlying formalism. In contrast, HProblog is a probabilistic extension of Prolog. There are differences in how each formalism models continuous variables. HRDNs, HProblog and CBLPs explicitly state the form of the distribution (e.g., a Gaussian) and its parameters (e.g., the mean and variance). In contrast, HMLNs express numeric variables through a set of soft constraints with a Gaussian penalty for diverging values. One notable difference between HRDNs and CBLPs is that CBLPs do not permit a discrete variable to have a continuous parent, whereas this is possible in HRDNs.

In terms of reasoning, HMLNs and HRDNs use approximate inference. Currently, HProblog only supports an exact inference procedure which involves partitioning the continuous probabilistic facts into admissible intervals. Scaling HProblog to large domains would

require the development of a suitable approximate inference algorithm. Inference in CBLPs can be split in two parts: logical inference and probabilistic inference. The former computes the support network for a query (i.e., a Bayesian network containing all relevant variables for the query). The latter applies off-the-shelf Bayesian network inference methods to the resulting support network.

There are significant differences in the level of support for learning in each formalism. Out of the four formalisms, HRDNs are the only one that support structure learning in hybrid domains. Like HRDNs, HMLNs and CBLPs have algorithms for parameter learning. Currently, HProblog does not support parameter learning.

7 Conclusions and future work

This paper addressed the problem of learning models from structured, relational data that contain both discrete and continuous variables. To the best of our knowledge, this is the first attempt to perform structure learning in a hybrid SRL setting. We introduced Hybrid Relational Dependency Networks (HRDNs), a novel extension of relational dependency networks that accommodate continuous variables and proposed an algorithm that automatically learns the structure of an HRDN from data. Empirically, we evaluated the benefit of incorporating continuous variables in a learned model on one synthetic and one real-world data set by considering two versions of each data set: one that contains both continuous and discrete variables, and one where each continuous variable is discretized prior to learning. We compared our proposed algorithm to two learners that work only on discrete data: a variant of our algorithm and LSM, the state-of-the-art MLN structure learner. We found that learning directly from the hybrid data resulted in more accurate learned models than learning from the discretized data.

One interesting direction for future work is to explore the suitability of modeling other continuous conditional distributions, next to the Gaussians considered in this paper. In principle, other density functions can be used given that we can calculate the value of the function at a point and that we can sample a value for a variable given the assignment to its parents. However, it is unclear how easy this is in practice for complex distributions, and whether issues could arise with sampling inconsistent HRDNs containing relational conditional dependencies. We would also like to extend our learning algorithm such that it could cope with missing data and model latent variables. Additionally, we would like to explore other penalty terms in the objective function such as a L1 penalty that has been used for learning propositional DNs (Dobra 2009; Meinshausen and Bühlmann 2006). Finally, we would like to evaluate our approach on more real-world domains.

Acknowledgments We would like to thank the anonymous reviewers and Guy Van den Broeck for their very helpful comments. Irma Ravkic is supported by the Research Fund KU Leuven (OT/11/051). Jan Ramon is supported by ERC-StG 240186, and the Research Fund KU Leuven (OT/11/051). Jesse Davis is partially supported by the Research Fund KU Leuven (OT/11/051), EU FP7 Marie Curie Career Integration Grant (#294068) and FWO-Vlaanderen (G.0356.12).

Appendix 1: Handcrafted model and learned hybrid models for the synthetic data

In this Appendix we compare the handcrafted and learned hybrid models for the synthetic data set. We present the learned dependencies for both setups: fixed domain size and

increasing domain size. For the former we show the learned dependencies when training on 16 interpretations, and for the latter we present the learned dependencies for the largest domain size (800 students, 125 courses and 125 professors).

Predicate declarations

$$range(\texttt{difficulty(C)}) = \{easy, med, hard\}$$
$$range(\texttt{satisfaction(S,C)}) = \{low, med, high\}$$
$$range(\texttt{grade(S,C)}) = \{low, med, high\}$$
$$range(\texttt{takes(S,C)}) = \{true, false\}$$
$$range(\texttt{teaches(P,C)}) = \{true, false\}$$
$$range(\texttt{friend(S,S1)}) = \{true, false\}$$
$$range(\texttt{nrhours(C)}) = [20.0, 180.0]$$
$$range(\texttt{intelligence(S)}) = [50.0, 180.0]$$
$$range(\texttt{ability(P)}) = [20.0, 100.0]$$

Handcrafted model

Below is the model we used to generate the synthetic data.

`difficulty(C)`	
`satisfaction(S,C)`	$\mathcal{F}_{\{S,C\}:\emptyset,grade(S,C),value}$, $\mathcal{F}_{\{C\}:teaches(P,C),ability(P),value}$
`grade(S,C)`	$\mathcal{F}_{\{S\}:\emptyset,intelligence(S),value}$, $\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$
`takes(S,C)`	$\mathcal{F}_{\{S\}:\emptyset,intelligence(S),value}$, $\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$
`teaches(P,C)`	$\mathcal{F}_{\{P\}:\emptyset,ability(P),value}$, $\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$
`friend(S,S1)`	$\mathcal{F}_{\{S,S1\}:\{takes(S,C),takes(S1,C)\},\emptyset,proportion}$
`nrhours(C)`	$\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$
`intelligence(S)`	$\mathcal{F}_{\{S\}:\emptyset,grade(S,C),mode}$
`ability(P)`	

Table 10 Local distributions used for the handcrafted model

Predicate	Local Distribution	Parameters
`difficulty/1`	Multinomial	(0.2, 0.4, 0.4)
`satisfaction/2`	Logistic Regression	`satisfaction(S,C)`=low →(-1.28,-0.07,0.028)
		`satisfaction(S,C)`=med→(0.5,-0.4,0.009)
`grade/2`	Logistic Regression	`grade(S,C)`=low →(-1.77,-0.04,1.75)
		`grade(S,C)`=med →(-2.18,0.003,0.75)
`takes/2`	Logistic Regression	`takes(S,C)`=true →(0.4,0.009,-0.607)
`teaches/2`	Logistic Regression	`teaches(P,C)`=true →(-0.089,-0.012,0.305)
`friend/2`	Logistic Regression	`friend(S,S1)`=true →(-0.08,1.5)
`nrhours/1`	Conditional Gaussian	`difficulty(C)`=easy → N(20,6)
		`difficulty(C)`=med → N(50,5)
		`difficulty(C)`=hard → N(80,6)
`intelligence/1`	Conditional Gaussian	`grade(S,C)`=low → N(60,5)
		`grade(S,C)`=med → N(90,7)
		`grade(S,C)`=high → N(110,5)
`ability/1`	Gaussian	N(70,10)

For reasons of reproducibility, we also provide the parameters for the dependencies for our handcrafted model. We will not do this for the learned models as there the parameters are of less interest. The parameters for the dependencies are given in Table 10. The probabilities of multinomial values are to be read in the order given in the predicate declaration. For the logistic regression of a dependency `P` | Parents(`P`), we use the notation `P` $= k \rightarrow (w_{k,0}, w_{k,\mathcal{F}_1}, \ldots, w_{k,\mathcal{F}_n})$ where $w_{k,0}$ represents the bias term, and $w_{k,\mathcal{F}_i}$ represents the ith feature's weight. The order of the feature parameters follows the order of the features in the dependencies. The parameters of conditional Gaussians are of the form $d \rightarrow N(\mu_d, \sigma_d)$, where d represents an instantiation of discrete parents, and $N(\mu_d, \sigma_d)$ gives the Gaussian distribution for that instantiation.

Learned model for a fixed domain size (16 training interpretations)

`difficulty(C)` | $\mathcal{F}_{\{C\}:\emptyset,nrhours(C),value}$

`satisfaction(S,C)` |

`grade(S,C)` | $\mathcal{F}_{\{S\}:\emptyset,intelligence(S),value}$, $\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$

`takes(S,C)` | $\mathcal{F}_{\{S,C\}:satisfaction(S,C)=low,\emptyset,proportion}$, $\mathcal{F}_{\{S,C\}:\emptyset,satisfaction(S,C),value}$

`teaches(P,C)` |

`friend(S,S1)`	$\mathcal{F}_{\{S\}:takes(S,C),\emptyset,proportion}$
`nrhours(C)`	$\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$
`intelligence(S)`	$\mathcal{F}_{\{S\}:\emptyset,grade(S,C),mode}$
`ability(P)`	

Learned model for a domain size of 800 students, 125 courses and 125 professors

`difficulty(C)`	$\mathcal{F}_{\{C\}:takes(S,C),\emptyset,proportion}$
`satisfaction(S,C)`	$\mathcal{F}_{\{S,C\}:\emptyset,grade(S,C),value}$, $\mathcal{F}_{\{C\}:teaches(P,C),ability(P),value}$
`grade(S,C)`	$\mathcal{F}_{\{S\}:\emptyset,intelligence(S),value}$, $\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$, $\mathcal{F}_{\{S,C\}:\emptyset,satisfaction(S,C),value}$
`takes(S,C)`	$\mathcal{F}_{\{S,C\}:\{satisfaction(S,C)=mid,friend(S,S1),takes(S1,C)\},\emptyset,proportion}$
`teaches(P,C)`	$\mathcal{F}_{\{P\}:\emptyset,ability(P),value}$
`friend(S,S1)`	$\mathcal{F}_{\{S\}:\{satisfaction(S,C)=low,grade(S,C)=high,takes(S,C)\},\emptyset,proportion}$
`nrhours(C)`	$\mathcal{F}_{\{C\}:\emptyset,difficulty(C),value}$
`intelligence(S)`	$\mathcal{F}_{\{S\}:\emptyset,grade(S,C),mode}$
`ability(P)`	

Appendix 2: PKDD'99 real-world financial data set

Table 11 Description of the predicates in the PKDD'99 financial data set

Predicate name	Description	Range
`clientAge(C,L)`	The age of client `C` at the moment of loan `L` origination	$\mathbb{R}$
`clientDistrict(C,D)`	Client `C` lives in district `D`	Boolean
`gender(C)`	The gender of client `C`	{m,f}
`hasAccount(C,A)`	Client `C` has an account `A`	Boolean
`avgSalary(D)`	The average salary in district `D`	$\mathbb{R}$
`ratUrbInhab(D)`	The ratio of urban inhabitants in district `D`	$\mathbb{R}$
`avgSumofW(A)`	The average sum of monthly withdrawals for account `A`	$\mathbb{R}$
`avgSumofCred(A)`	The average sum of monthly credits for account `A`	$\mathbb{R}$
`stdOfW(A)`	The standard deviation of monthly withdrawals for account `A`	$\mathbb{R}$
`stdOfCred(A)`	The standard deviation of monthly credits for account `A`	$\mathbb{R}$
`freq(A)`	The frequency of statement issuance for account `A`	{i,d,m}
`avgNrW(A)`	The average number of withdrawals per a month for account `A`	$\mathbb{R}$
`hasLoan(A,L)`	Account `A` has loan `L`	Boolean
`loanAmount(L)`	The amount of loan `L`	$\mathbb{R}$
`loanStatus(L)`	The status of loan `L`	{a,b,c,d}
`monthlyPayments(L)`	The monthly payment amount for loan `L`	$\mathbb{R}$

Appendix 3: Detailed results for all domains

In this Appendix we present detailed results on per predicate WPLLs for all domains used in our experiments.

Results on synthetic data

Tables 12,13, 14, 15 and 16 show the test set per randvar WPLLs for each predicate when varying the number of training interpretations. Tables 17, 18, 19 and 20 show the test set per randvar WPLLs for each predicate when varying the domain size of the training interpretations. In both cases, the WPLLs are averaged over all ten runs.

Results on the PKDD'99 financial data set

Tables 21 and 22 contain per randvar WPLLs for all learners applied on the PKDD'99 financial data set. All the WPLLs represent an average value over ten-fold cross-validation.

Table 12 The per randvar WPLL for each predicate on the synthetic data when training on one interpretation

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.57**	−5.46	−6.30	−4.90	−6.25	−4.81	−6.11	−5.41	−6.05
difficulty/1	**−0.06**	−0.83	−1.59	−0.27	−1.59	−0.28	−1.59	−0.17	−1.59
ability/1	**−5.34**	−5.95	−5.95	−5.70	−5.70	−5.71	−5.72	−5.72	−5.67
intelligence/1	**−4.89**	−5.71	−7.24	−5.34	−6.09	−5.21	−6.01	−5.39	−5.94
grade/2	−1.49	−1.51	−1.53	**−1.48**	−1.53	**−1.48**	−1.53	**1.48**	−1.53
satisfaction/2	−1.55	−1.55	**−1.54**	−1.56	**−1.54**	−1.56	**−1.54**	−1.55	**−1.54**
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**
friend/2	**−0.14**	**−0.14**	−0.15	−0.15	−0.15	−0.15	−0.15	−0.15	−0.15
teaches/2	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**
Total WPLL	**−18.22**	−21.33	−24.45	−19.53	−23.00	−19.34	−22.79	−20.03	−22.62

The best results are in bold

Table 13 The per randvar WPLL for each predicate on the synthetic data when training on two interpretations

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.43**	−5.41	−6.27	−4.82	−6.20	−4.61	−6.09	−5.01	−6.14
difficulty/1	−0.18	−0.69	−1.56	−0.28	−1.56	−0.08	−1.56	**−0.10**	−1.44
ability/1	**−5.33**	−5.93	−5.96	−5.66	−5.66	−5.67	−5.67	−5.66	−5.72
intelligence/1	**−4.87**	−5.71	−7.18	−5.21	−6.08	−5.09	−5.99	−5.01	−5.96
grade/2	−1.50	−1.51	−1.53	**−1.46**	−1.53	−1.48	−1.53	−1.47	−1.53
satisfaction/2	−1.55	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	**−1.54**
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**
friend/2	**−0.15**	−0.16	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**
teaches/2	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**	**−0.14**
Total WPLL	**−18.16**	−21.10	−24.34	−19.27	−22.86	−18.77	−22.67	−19.11	−22.62

The best results are in bold

Table 14 The per randvar WPLL for each predicate on the synthetic data when training on four interpretations

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.37**	−5.41	−6.07	−4.82	−5.62	−4.58	−6.07	−4.73	−5.44
difficulty/1	**−0.01**	−0.66	−1.31	−0.26	−0.95	−0.06	−0.90	−0.08	−0.85
ability/1	**−5.31**	−5.92	−5.92	−5.64	−5.64	−5.65	−5.68	−5.64	−5.68
intelligence/1	**−4.85**	−5.71	−6.12	−5.19	−6.08	−4.98	−5.98	−4.89	−5.94

Table 14 continued

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
grade/2	−1.50	−1.51	−1.53	**−1.45**	−1.53	**−1.45**	−1.53	−1.46	−1.53
satisfaction/2	−1.55	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	**−1.54**
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**
friend/2	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**	**−0.15**
teaches/2	**−0.10**	**−0.14**	**−0.14**	**−0.14**	**−0.10**	**−0.14**	**−0.14**	**−0.14**	**−0.14**
Total WPLL	**−17.89**	−21.06	−20.52	−19.20	−21.65	−18.56	−22.00	−18.64	−21.27

The best results are in bold

Table 15 The per randvar WPLL for each predicate on the synthetic data when training on eight interpretations

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.35**	−5.41	OoM	−4.82	OoM	−4.58	OoM	−4.73	OoM
difficulty/1	**−0.02**	−0.67	OoM	−0.18	OoM	−0.06	OoM	−0.09	OoM
ability/1	**−5.31**	−5.92	OoM	−5.63	OoM	−5.65	OoM	−5.63	OoM
intelligence/1	**−4.86**	−5.71	OoM	−5.18	OoM	−4.97	OoM	−4.88	OoM
grade/2	−1.49	−1.51	OoM	−1.46	OoM	−1.46	OoM	**−1.45**	OoM
satisfaction/2	**−1.55**	**−1.55**	OoM	**−1.55**	OoM	**−1.55**	OoM	**−1.55**	OoM
takes/2	**−0.00**	**−0.00**	OoM	**−0.00**	OoM	**−0.00**	OoM	**−0.00**	OoM
friend/2	**−0.15**	**−0.15**	OoM	**−0.15**	OoM	**−0.15**	OoM	**−0.15**	OoM
teaches/2	−0.14	**−0.14**	OoM	**−0.14**	OoM	**−0.14**	OoM	**−0.14**	OoM
Total WPLL	**−17.87**	−21.06	OoM	−19.12	OoM	−18.55	OoM	−18.46	OoM

The best results are in bold, and OoM denotes out of memory

Table 16 The per randvar WPLL for each predicate on the synthetic data when training on 16 interpretations

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.35**	−5.41	OoM	−4.79	OoM	−4.57	OoM	−4.50	OoM
difficulty/1	**−0.02**	−0.66	OoM	−0.15	OoM	−0.05	OoM	−0.04	OoM
ability/1	**−5.31**	−5.91	OoM	−5.63	OoM	−5.64	OoM	−5.61	OoM
intelligence/1	**−4.83**	−5.71	OoM	−5.18	OoM	−4.96	OoM	−4.87	OoM
grade/2	−1.49	−1.51	OoM	−1.46	OoM	−1.46	OoM	**−1.45**	OoM
satisfaction/2	**−1.55**	**−1.55**	OoM	**−1.55**	OoM	**−1.55**	OoM	**−1.55**	OoM
takes/2	**−0.00**	**−0.00**	OoM	**−0.00**	OoM	**−0.00**	OoM	**−0.00**	OoM
friend/2	**−0.15**	**−0.15**	OoM	**−0.15**	OoM	**−0.15**	OoM	**−0.15**	OoM

Table 16 continued

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
teaches/2	**−0.14**	**−0.14**	OoM	**−0.14**	OoM	**−0.14**	OoM	**−0.14**	OoM
Total WPLL	**−17.85**	−21.05	OoM	−19.04	OoM	−18.52	OoM	−18.30	OoM

The best results are in bold, and OoM denotes out of memory

Table 17 The per randvar WPLL for each predicate on the synthetic data consisting of 100 students, 50 courses and 50 professors

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.66**	−5.35	−6.30	−4.94	−6.25	−5.04	−6.11	−5.63	−6.05
difficulty/1	**−0.07**	−0.91	−1.59	−0.45	−1.59	−0.28	−1.59	−0.34	−1.59
ability/1	−5.35	−5.46	−5.95	−5.21	−5.70	**−5.15**	−5.72	−5.18	−5.67
intelligence/1	**−4.73**	−5.54	−7.24	−5.06	−6.09	−4.86	−6.01	−4.93	−5.94
grade/2	**−1.51**	**−1.51**	−1.53	**−1.51**	−1.53	**−1.51**	−1.53	−1.52	−1.53
satisfaction/2	−1.55	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	**−1.54**
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**
friend/2	−0.16	−0.16	**−0.15**	−0.16	**−0.15**	−0.16	**−0.15**	−0.16	**−0.15**
teaches/2	**−0.07**	**−0.07**	−0.14	**−0.07**	−0.14	**−0.07**	−0.14	**−0.07**	−0.14
Total WPLL	**−18.11**	−20.56	−24.45	−18.95	−23.00	−18.62	−22.79	−19.39	−22.62

The best results are in bold

Table 18 The per randvar WPLL for each predicate on the synthetic data consisting of 200 students, 75 courses and 75 professors

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.50**	−5.28	−6.11	−4.69	−6.08	−4.70	−6.11	−4.76	−6.13
difficulty/1	**−0.05**	−0.71	−1.57	−0.18	−1.57	−0.24	−1.59	−0.14	−1.59
ability/1	−5.34	−5.60	−5.60	−5.29	−5.29	**−5.26**	−5.58	**−5.26**	−5.62
intelligence/1	**−4.70**	−5.61	−6.00	−5.11	−5.99	−4.77	−6.11	−4.77	−6.14
grade/2	−1.48	−1.50	−1.54	**−1.46**	−1.54	−1.47	−1.58	−1.47	−1.59
satisfaction/2	−1.55	−1.55	**−1.54**	−1.55	**−1.54**	−1.55	−1.59	−1.55	−1.59
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	−1.00	**−0.00**	−1.00
friend/2	**−0.16**	**−0.16**	**−0.16**	**−0.16**	−0.24	**−0.16**	−1.00	**−0.16**	−1.00
teaches/2	**−0.07**	**−0.07**	**−0.07**	**−0.07**	−0.91	−0.08	−1.00	**−0.07**	−1.00
Total WPLL	**−17.84**	−20.47	−22.25	−18.50	−23.15	−18.22	−25.55	−18.17	−25.64

The best results are in bold

Table 19 The per randvar WPLL for each predicate on the synthetic data consisting of 400 students, 100 courses and 100 professors

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.52**	−5.32	−6.14	−4.72	−6.09	−4.51	−5.96	−4.76	−5.89
difficulty/1	**−0.02**	−0.70	−1.55	−0.17	−1.55	−0.12	−1.55	−0.10	−1.55
ability/1	−5.33	−5.53	−5.53	−5.28	−5.28	−5.21	**−5.18**	−5.22	−5.19
intelligence/1	**−4.67**	−5.62	−6.01	−5.10	−5.98	−4.79	−5.84	−4.74	−5.78
grade/2	−1.46	−1.48	−1.53	**−1.45**	−1.54	**−1.45**	−1.54	−1.46	−1.53
satisfaction/2	**−1.54**	**−1.54**	**−1.54**	**−1.54**	**−1.54**	**−1.54**	**−1.54**	**−1.54**	**−1.54**
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**
friend/2	**−0.16**	**−0.16**	**−0.16**	**−0.16**	**−0.16**	**−0.16**	**−0.16**	**−0.16**	**−0.16**
teaches/2	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**
Total WPLL	**−17.78**	−20.42	−22.53	−18.48	−22.20	−17.86	−21.83	−18.05	−21.71

The best results are in bold

Table 20 The per randvar WPLL for each predicate on the synthetic data consisting of 800 students, 125 courses and 125 professors

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
nrhours/1	**−4.48**	−5.33	−6.18	−4.78	−5.96	−4.54	−5.99	−4.58	−5.92
difficulty/1	**−0.02**	−0.66	−1.53	−0.17	−1.55	−0.09	−1.53	−0.12	−1.53
ability/1	**−5.34**	−5.67	−5.67	−5.31	−5.18	−5.26	−5.26	−5.24	−5.24
intelligence/1	**−4.66**	−5.65	−6.04	−5.13	−5.84	−4.79	−5.84	−4.73	−5.79
grade/2	−1.45	−1.47	−1.54	**−1.44**	−1.54	**−1.44**	−1.53	**−1.44**	−1.53
satisfaction/2	−1.54	−1.54	−1.54	−1.54	−1.54	**−1.53**	−1.54	**−1.53**	−1.54
takes/2	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**	**−0.00**
friend/2	**−0.15**	**−0.15**	**−0.15**	−0.15	−0.16	**−0.15**	**−0.15**	−0.16	**−0.15**
teaches/2	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**	**−0.07**
Total WPLL	**−17.72**	−20.54	−22.72	−18.60	−21.83	−17.87	−21.92	−17.86	−21.79

The best results are in bold

Table 21 The per randvar WPLL for each predicate on the PKDD'99 financial data set. The best results are in bold

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
avgNrWith/1	−4.60	−4.81	−4.83	−4.61	−4.75	**−4.56**	−4.85	−4.60	−5.05
avgSalary/1	−11.24	−11.35	−11.44	−10.96	−11.25	−10.82	−11.39	**−10.68**	−11.65
avgSumofCred/1	**−14.54**	−17.31	−17.58	−16.44	−17.11	−16.01	−16.34	−15.70	−16.88
avgSumOfW/1	**−14.39**	−17.22	−17.43	−16.34	−17.00	−15.85	−16.26	−15.54	−16.81
clientAge/2	−5.77	−5.72	−5.71	−5.67	−5.70	−5.60	−5.74	**−5.59**	−5.73
clientDistrict/2	**−0.09**	**−0.09**	−0.11	**−0.09**	**−0.09**	**−0.09**	−0.12	**−0.09**	−0.15
freq/1	**−0.38**	−0.41	−0.45	−0.39	−0.42	**−0.38**	−0.50	−0.39	−0.45
gender/1	**−1.00**	**−1.00**	**−1.00**	**−1.00**	**−1.00**	**−1.00**	**−1.00**	**−1.00**	**−1.00**
hasAccount/2	**−0.02**	**−0.02**	−0.03	**−0.02**	**−0.02**	**−0.02**	−0.03	**−0.02**	−0.03
hasLoan/2	**−0.01**	**−0.01**	−0.03	−0.07	−0.04	−0.06	−0.03	−0.13	−0.03
loanAmount/1	−18.26	−18.48	−18.54	**−18.17**	−18.40	−18.19	−18.62	−18.35	−18.80
loanStatus/1	**−1.32**	−1.42	−1.48	−1.40	−1.43	−1.40	−1.45	−1.50	−1.48
monthlyPayments/1	−12.70	−12.96	−13.02	**−12.69**	−12.89	−12.91	−13.09	−12.80	−13.15
ratUrbInhab/1	−5.74	−5.83	−5.93	−5.56	−5.77	−5.40	−5.86	**−5.29**	−5.91
stdOfCred/1	**−14.05**	−15.65	−15.90	−14.98	−15.51	−14.67	−15.51	−14.34	−16.10
stdOfW/1	**−13.81**	−15.36	−15.56	−14.62	−15.18	−14.32	−15.06	−14.16	−15.66
Total WPLL	**−117.93**	−127.64	−129.02	−122.99	−126.55	−121.27	−125.83	−120.17	−128.89

Table 22 The per randvar WPLL of the two variants of the handcrafted models, LR+LinR and LR+MP5, compared to LLM-H on the hybrid data for the PKDD'99 financial data set

Predicate	LR+LinR	LR+MP5	LLM-H
avgNrWith/1	−0.96	−0.96	**−0.09**
avgSalary/1	**−1.00**	**−1.00**	**−1.00**
avgSumofCred/1	−1.00	−1.00	**−0.02**
avgSumOfW/1	**−0.36**	**−0.36**	−0.38
clientAge/2	−0.89	−0.89	**−0.01**
clientDistrict/2	**−1.17**	**−1.17**	−1.32
freq/1	−5.78	**−5.77**	**−5.77**
gender/1	−11.18	**−10.98**	−11.24
hasAccount/2	**−5.74**	**−5.74**	**−5.74**
hasLoan/2	**−14.35**	−14.98	−14.39
loanAmount/1	**−14.51**	−15.08	−14.54
loanStatus/1	**−13.81**	−13.82	**−13.81**
monthlyPayments/1	**−14.05**	−13.89	**−14.05**
ratUrbInhab/1	−4.31	**−4.10**	−4.60
stdOfCred/1	**−18.14**	−18.15	−18.26
stdOfW/1	**−12.54**	−12.61	−12.70
Total WPLL	−119.79	−120.22	**−117.93**

LR+LinR uses logistic regression for discrete predicates and linear regression for continuous predicates, and LR+MP5 uses logistic regression for discrete predicates and regression trees for continuous predicates. The best results are in bold

Appendix 4: Features used for propositional learners

In order to compare our structure learning algorithm to propositional learners on the PKDD'99 financial data set, we handcrafted a number of features for each of the 16 predicates. Each feature predicts a property of an object by using some other properties of that object.

Predicates with discrete range

`clientDistrict(C,D)`	$\mathcal{F}_{\{C\}:\emptyset,gender(C),value}$ $\mathcal{F}_{\{C\}:\emptyset,avgSalary(D),value}$ $\mathcal{F}_{\{C\}:\emptyset,ratUrbInhab(D),value}$

`gender(C)`	$\mathcal{F}_{\{C\}:\emptyset,hasAccount(A,C),exists}$

`hasAccount(C,A)`	$\mathcal{F}_{\{C\}:\emptyset,gender(C),value}$ $\mathcal{F}_{\{A\}:\emptyset,hasLoan(A,L),exists}$ $\mathcal{F}_{\{A\}:\emptyset,freq(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgNrWith(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfCred(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfCred(A),value}$

`freq(A)`	$\mathcal{F}_{\{A\}:\emptyset,avgNrWith(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfCred(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfCred(A),value}$

`hasLoan(A,L)`	$\mathcal{F}_{\{L\}:\emptyset,loanAmount(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,loanStatus(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,monthlyPayments(L),value}$ $\mathcal{F}_{\{A\}:\emptyset,freq(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgNrWith(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfCred(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfCred(A),value}$

`loanStatus(L)`	$\mathcal{F}_{\{L\}:\emptyset,loanAmount(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,loanStatus(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,monthlyPayments(L),value}$

Predicates with continuous range

The following are the features we used for predicates with a continuous range. Note that the predicates `avgSumOfW/1`, `avgSumOfCred/1`, `stdOfW/1` and `stdOfCred/1` have similar structure as the features for `avgNrWith(A)`. To save space we will only show the features we used for `avgNrWith(A)`. The full feature set is in the online appendix on http://dtai.cs.kuleuven.be/ml/systems/llm.

avgSalary(D)	$\mathcal{F}_{\{D\}:\emptyset,clientDistrict(C,D),proportion}$ $\mathcal{F}_{\{D\}:\emptyset,ratUrbInhab(D),value}$
loanAmount(L)	$\mathcal{F}_{\{L\}:\emptyset,loanStatus(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,monthlyPayments(L),value}$
monthlyPayments(L)	$\mathcal{F}_{\{L\}:\emptyset,loanStatus(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,loanAmount(L),value}$
avgNrWith(A)	$\mathcal{F}_{\{A\}:\emptyset,freq(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,avgSumOfCred(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfW(A),value}$ $\mathcal{F}_{\{A\}:\emptyset,stdOfCred(A),value}$
ratUrbInhab(D)	$\mathcal{F}_{\{D\}:\emptyset,avgSalary(D),value}$ $\mathcal{F}_{\{D\}:\emptyset,clientDistrict(C,D),proportion}$
clientAge(C,L)	$\mathcal{F}_{\{C\}:\emptyset,gender(C),value}$ $\mathcal{F}_{\{L\}:\emptyset,loanAmount(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,loanStatus(L),value}$ $\mathcal{F}_{\{L\}:\emptyset,monthlyPayments(L),value}$

References

Berka, P. (1999). PKDD'99 Discovery challenge: http://lisp.vse.cz/pkdd99/Challenge/.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society Series B (Methodological)*, *36*, 192–236.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. New York, NY, USA: Oxford University Press Inc.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Choi, J., Amir, E., Hill, D.J. (2010). Lifted inference for relational continuous models. In: UAI'10: Proceedings of the twenty-sixth conference on uncertainty in artificial intelligence, pp. 126–134.

Cobb, B., Rumí, R., & Salmerón, A. (2007). Bayesian network models with discrete and continuous variables. *Advances in probabilistic graphical models* (Vol. 214, pp. 81–102). Berlin, Heidelberg: Springer.

Dobra, A. (2009). Variable selection and dependency networks for genomewide data. *Biostatistics (Oxford, England)*, *10*, 621–639.

Domingos, P., & Provost, F. (2000). *Well-trained PETs: Improving probability estimation trees. CDER Working Paper, Stern School of Business*. New York, NY: New York University.

Fierens, D., Blockeel, H., Bruynooghe, M., & Ramon, J. (2005). Logical Bayesian networks and their relation to other probabilistic logical models. In: *Proceedings of the 15th international conference on inductive logic programming*, (Vol. 3625, pp. 121–135) Berlin: Springer.

Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge: The MIT press.

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001) Learning probabilistic relational models. In: *Relational Data Mining*. Springer, Berlin.

Guo, Y., & Gu, S. (2011). Multi-label classification using conditional dependency networks. In: *Proceedings of the twenty-second international joint conference on artificial intelligence*, IJCAI'11 (Vol. 2, pp. 1300–1305).

Gutmann, B., Jaeger, M., & De Raedt, L. (2011). Extending ProbLog with continuous distributions. In: Inductive Logic Programming (pp. 76–91) Berlin: Springer.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, *11*(1).

Heckerman, D., Chickering, D.M., Meek, C., Rounthwaite, R., & Kadie, C. (2001). Dependency networks for inference, collaborative filtering, and data visualization. *Journal ofMachine LearningResearch*, 49–75.

Kersting, K., & De Raedt, L. (2001). Adaptive Bayesian logic programs. In: *Inductive Logic Programming* (pp. 104–117). Berlin: Springer.

Kok, S., & Domingos, P. (2005). Learning the Structure of Markov Logic Networks. In: *Proceedings of the 22Nd international conference on machine learning*, ICML '05, pp. 441–448.

Kok, S., & Domingos, P. (2010). Learning Markov logic networks using structural motifs. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 551–558.

Koller, D., Lerner, U., & Angelov, D. (1999). A general algorithm for approximate inference and its application to hybrid Bayes nets. In: *Proceedings of the fifteenth conference on uncertainty in artificial intelligence*, pp. 324–333.

Kuželka, O., Szabóová, A., Holec, M., & Železný, F. (2011). Gaussian logic for predictive classification. *Machine learning and knowledge discovery in databases, Lecture Notes in Computer Science* (Vol. 6912, pp. 277–292). Berlin, Heidelberg: Springer.

Lauritzen, S. L. (1992). Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, *87*, 1098–1108.

Lauritzen, S. L., & Jensen, F. (2001). Stable local computation with conditional Gaussian distributions. *Statistics and Computing*, *11*, 191–203.

Meinshausen, N., & Bühlmann, P. (2006). High dimensional graphs and variable selection with the Lasso. *Annals of statistics*, *34*, 1436–1462.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Moral, S., Rumi, R., & Salmerón, A. (2001). Mixtures of truncated exponentials in hybrid Bayesian networks. *Symbolic and quantitative approaches to reasoning with uncertainty, Lecture Notes in Computer Science* (Vol. 2143, pp. 156–167). Berlin, Heidelberg: Springer.

Murphy, K.P. (1998). *Inference and learning in hybrid Bayesian networks*. Tech. Rept. UCB/CSD-98-990, U.C. Berkeley, CA.

Narman, P., Buschle, M., Konig, J., & Johnson, P. (2010). Hybrid probabilistic relational models for system quality analysis. In: *Proceedings of the 2010 14th IEEE international enterprise distributed object computing conference*, pp. 57–66.

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., & Shavlik, J. (2012). Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, *86*, 25–56.

Neville, J., & Jensen, D. (2007). Relational dependency networks. *Journal of Machine Learning Research*.

Neville, J., Jensen, D., Friedland, L., & Hay, M. (2003). Learning relational probability trees. In: *Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining*, ACM Press, pp. 625–630.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine learning*, *62*, 107–136.

Romero, V., Rumí, R., & Salmerón, A. (2006). Learning hybrid Bayesian networks using mixtures of truncated exponentials. *International Journal of Approximate Reasoning*, *42*, 54–68.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 461–464.

Teso, S., Sebastiani, R., & Passerini, A. (2013). *Hybrid SRL with optimization modulo theories*. In *2013 NIPS Workshop on Constructive Machine Learning*. Lake Tahoe, Nevada, USA.

Wang, J., & Domingos, P. (2008). Hybrid Markov logic networks. *In: Proceedings of the 23rd national conference on Artificial intelligence*, Vol. 2, pp. 1106–1111.

Yuan, C., & Druzdzel, M.J. (2007). Importance sampling for general hybrid Bayesian networks. In: Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS-07), Journal of Machine Learning Research - Proceedings Track, vol 2, pp. 652–659.