Learning Probabilistic Logic Programs over Continuous Data*

Stefanie Speichert[1] and Vaishak Belle[1,2]

[1] University of Edinburgh, UK
[2] Alan Turing Institute, UK

Abstract. The field of statistical relational learning aims at unifying logic and probability to reason and learn from data. Perhaps the most successful paradigm in the field is probabilistic logic programming (PLP): the enabling of stochastic primitives in logic programming. While many systems offer inference capabilities, the more significant challenge is that of learning meaningful and interpretable symbolic representations from data. In that regard, inductive logic programming and related techniques have paved much of the way for the last few decades, but a major limitation of this exciting landscape is that only discrete features and distributions are handled. Many disciplines express phenomena in terms of continuous models.

In this paper, we propose a new computational framework for inducing probabilistic logic programs over continuous and mixed discrete-continuous data. Most significantly, we show how to learn these programs while making no assumption about the true underlying density. Our experiments show the promise of the proposed framework.

1 Introduction

The field of statistical relational learning (SRL) aims at unifying logic and probability to reason and learn from relational data. Perhaps the most successful paradigm here is probabilistic (logic) programming (PLP): the enabling of stochastic primitives in (logic) programming, which is now increasingly seen to provide a declarative basis to complex machine learning applications [20, 14].

While many PLP systems offer inference capabilities [4, 14, 31], the more difficult task is that of learning meaningful and interpretable symbolic representations from data. *Parameter learning* attempts to obtain the probabilities of atoms from observational traces (e.g., number of heads observed in a sequence of coin tosses). *Structure learning* goes a step further and attempts to learn deterministic or probabilistic rules (that is, logic programs) from data. In that regard, *inductive logic programming (ILP)* and first-order logic (FOL) rule learning have paved much of the way for the last few decades [32, 38, 13], with important applications for Web and biomedical data [44, 12], among others.

* This work is partly supported by the EPSRC grant *Towards Explainable and Robust Statistical AI: A Symbolic Approach.*

A major limitation of this exciting landscape is that only discrete features and distributions are handled. This is somewhat surprising, as many disciplines express phenomena in terms of continuous models. The heart of the matter is that *inference is already very challenging*. Indeed, inference schemes often assume parametric families (e.g., Gaussians) [28, 22], or are approximate offering asymptotic guarantees that only ensure correctness in the limit [33]. The learning of models is analogously restricted, as learning uses inference as a sub-routine [25]. Consequently, there is very little treatment on continuous distributions, both on the inference and learning fronts. (We discuss some notable exceptions in the penultimate section.)

In this work, we study the problem of parameter and structure learning for PLPs over continuous and mixed discrete-continuous data. Critically, we will not assume that these distributions are taken from known parametric families. To the best of our knowledge, this is the first such attempt and we hope it will make probabilistic knowledge representation systems more widely applicable for uncertain hybrid data. In particular, we propose a computational framework for inducing probabilistic logic programs over continuous and mixed discrete-continuous data. Rather than needing to define a new learning paradigm, we show that it is possible to leverage existing structure learners by appealing to the formulation of *weighted model integration* (WMI). WMI has been proposed recently as a computational abstraction for inference in discrete-continuous domains [6]. It generalizes *weighted model counting* [9], which is used for inference in ProbLog [18], for example. The resulting system is then a new dialect of the popular PLP language ProbLog [18], which we call *WMIProbLog*. On the one hand, WMIProbLog supports learning with hybrid distributions, and on the other, it enables *efficient exact inference*. We remark that our framework is very general, and could easily be adapted to other PLP languages.

2 Framework

Our framework consists of a generic weight and structure learner for hybrid data (i.e., data containing continuous and discrete attributes), to yield weighted hybrid atoms and thus, hybrid programs. As mentioned above, our approach piggybacks on existing structure learners, including techniques for density estimation that impose no restrictions on the true density that generated the data.

Overall, our algorithmic pipeline is as follows:

1. Intervals that best represent the data are constructed and polynomial weights for these intervals are learned, based on algorithms for *basis splines*. Basically, the attributes are split into mutually exclusive intervals. Then, a piecewise polynomial (PP) approximation of the density is learned by the algorithm with the understanding that each piece represents the probability density function (PDF) for that interval. The constructed intervals lead to "invented" predicates.
2. Relations between these atoms are constructed as clauses, subsequently yielding hybrid probabilistic programs. That is, rules to relate the newly invented atoms are induced by adapting the inputs of two discrete probabilistic FOL rule learners.
3. Finally, the atoms, weights and rules are combined in a PLP language equipped with exact inference via WMI, which we call *WMIProbLog*. Exact inference is

made possible by means of the PP representation, which admits efficient integration [3]. This is a major contribution in itself because most SRL languages with continuous distributions only support approximate inference over weak asymptotic guarantees [34].

The organization of the paper is as follows. We discuss the technical background for realizing the above steps, before turning to empirical evaluations, related work and conclusions.

2.1 Learning Weighted Atoms

We present a fully *unsupervised approach* to jointly learn intervals and their piecewise polynomial approximate densities from data. Our objective is defined as follows:

Definition 1 (Weighted Atom Construction) *Given:*

1. *a finite set of numeric data points* $E = \{x_1, ..., x_n\}$ *where* $t(x_j)$ *is a ground fact for the predicate* t.[3]
2. *A partition scheme* $E_1, ..., E_l$ *that partitions* E *into* l *parts where* $\cup_i E_i = E$, $E_i \cap E_j = \emptyset$ *and* $E_i \subseteq E$.
3. *An unknown distribution for each partition* i*, that is,* $f_i : E_i \rightarrow \mathbb{R}_{\geq 0}$ *such that* $\sum_{i=1}^{l} \int_{E_i} f_i = 1$.
4. *A hypothesis space* $\mathbf{H}$ *that consists of piecewise polynomials where* $H \in \mathbf{H}$ *is of the form* $\langle h_1, ...h_l \rangle$*, such that* $\sum_{i=1}^{l} \int_{E_i} h_i = 1$.
5. *A loss function measuring the loss/granularity tradeoff for a candidate density-estimator* h_i*:* $loss(h_i, f_i)$ *for each* i.[4]

Find: *A hypothesis* H *such that* $loss(f_i, h_i)$ *is minimized for each* i.

The goal of this learner is to find an optimal piecewise polynomial density estimation that is as close to the unknown underlying distribution as possible. As a result, we obtain new weighted atoms of the form: $\mathtt{h_i} :: \mathtt{t_i(X)} \leftarrow \mathtt{t(X)}, \mathtt{E_i(X)}$ for each i. For example, we might learn the following for the $\mathtt{height(X)}$ predicate over the $[60, 91]$ region: $\mathtt{X^2/3} :: \mathtt{height_60_91(X)} \leftarrow \mathtt{height(X)}, \mathtt{X} \in [\mathtt{60}, \mathtt{91}]$. This says that the atoms whose groundings lay between 60 and 91 such as $\mathtt{height_60_91(60.4)}$ have a probabilistic density given by the polynomial $\mathtt{X^2/3}$. We provide an algorithm that induces such weighted predicates. It involves two steps: *partitioning* and *density estimation*.

Step 1: Partitioning Partitioning refers to dividing the range of a continuous variable into l mutually exclusive intervals according to selected criteria. Our motivation is to capture complex continuous data by offloading the continuity to the weight functions

[3] Imagine, for example, the predicate $\mathtt{height(X)}$ with examples such as $\mathtt{height(60.4)}, \ldots, \mathtt{height(91.1)}, \ldots, \mathtt{height(124.6)}$.

[4] That is, we may want to penalise very granular representations that are defined over a large number of intervals and polynomials of a high degree. So, we would like to minimise the loss, but prefer simpler representations over granular ones.

so as to yield meaningful hybrid atoms. Suppose X is a real-valued variable – logically, think of the argument in $height(X)$ – and suppose $x_{min}, \ldots, x_{max}$ are the data points we observe. Partitioning yields $I = \{E_1, ..., E_l\}$ with $E_i = [c_{i-1}, c_i]$. The set of cutpoints $C = \{c_0, ..., c_l\}$ determine the interval such that $c_{i-1} < c_i$, $i \in \{1, ..., l\}$, $c_0 = x_{min}$ and $c_l = x_{max}$. The partiton step, therefore, defines intervals over the domain $\Omega = [x_{min}, x_{max}]$.

To restrict the interval search we chose to limit ourselves to two simple but effective schemes: *equal-width* and *equal-frequency* binning [16]. Both methods are regulated by the same parameter l that determines the number of bins that is considered by the algorithm.

Step 2: Density Estimation To enable exact inference over hybrid queries, we choose piecewise polynomial functions to approximate the density. The main advantage is that such PP densities can be learned from data without *any* prior information about the density while still capturing the true density very closely [46]. This section introduces the problem of approximating densities through PPs and shows how to learn the weighted atoms from data.

Definition 2 *A piecewise function over a real-valued variable x is defined over l pieces as:*

$$\boldsymbol{\delta}(X) = \begin{cases} 0 & x < c_0 \\ \delta_1(x) & c_0 \leq x \leq c_1 \\ \ldots & \\ \delta_l(x) & c_{l-1} \leq x \leq c_l \\ 0 & x > c_l \end{cases}$$

where the intervals (expressed using cutpoints) are mutually exclusive, and each $\delta_i(x)$ is a polynomial with the same maximum polynomial order k of the form: $\delta_i(x) = b_0^i + b_1^i * x + \ldots + b_k^i * x^k$. In order for $\boldsymbol{\delta}(x)$ to form a valid density, the function must be non-negative and integrate to 1: $\sum_{i=1}^{l} \int_{c_{i-1}}^{c_i} \delta_i(x)dx = 1$.

Our overall objective is to learn PP weights for each of the intervals. That is, suppose $f : \mathbb{R} \rightarrow \mathbb{R}_+$ is the density of an unknown distribution. We would like to find a candidate PP hypothesis h, defined over l intervals such that the polynomial for each interval is of degree at most d. Let $P_{l,d}$ be the class of l-piecewise degree-d polynomials. Then, we wish to find m hypotheses $h_i^1, ..., h_i^m \in P_{l,d}$ for each interval i and then select one that maximises the likelihood of the observed data. Since we also do not know the optimal degree to choose from, we let the same likelihood criterion also determine the best degree. For this, we leverage known techniques for learning polynomials [29] based on *basis splines* (B-Splines) [11]. B-Splines form a basis in the piecewise polynomial space. By considering linear combinations of splines, more complex polynomials can be obtained. The combination is adjusted by a set of mixing coefficients. By imposing constraints on these coefficients, polynomials can form a (valid) density: they are continuous, non-negative, differentiable and integrate to 1.

The parameters, the degree of the polynomial d and the number of pieces l are estimated by the Bayesian information criterion [45, 29]:

$$BIC(x, \boldsymbol{\delta}(x)) = \mathcal{L}(x \mid \boldsymbol{\delta}(x)) - \frac{log(|X|)}{2}$$

where $\mathcal{L}$ denotes the log-likelihood measure. The BIC score is known to be robust and avoids overfitting the model parameters. Moreover, in our empirical results, it was seen to favor smaller polynomial ranks which achieve the desired loss/granularity tradeoff discussed earlier.

Our algorithm is as follows:

1. **Input:** data points for real-valued variable X, and user-specified values for maximum number of intervals allowed ($maxSize = 40$), and maximum degree for polynomials allowed ($maxOrder = 9$).
2. Sort the points for X in ascending order, initialize B (the *best BIC score*) to $-\infty$, and S (the *best polynomial structure*) to {}.
3. Loop the following steps for $2 \leq l \leq maxSize$ intervals.
4. Loop the following steps for partition choices $d \in \{$*equal width, equal frequency*$\}$.
5. Obtain cutpoint set C from (X, d).
6. Loop the following steps for polynomial degree $1 \leq k \leq maxOrder$.
7. Define temporary variables s and b, the former containing a PP derived from (C, k), and the latter the BIC score derived from s.
8. If b exceeds B, then let $B = b$, and $S = s$.
9. Return to (6) with the next iteration of k until completion, following which return to (4) with the next iteration of d until completion, following which return to (3) with the next iteration of l until completion.
10. **Output:** (S, B).

Theorem 1 *The above algorithm realises Definition 1 with BIC measuring the loss and a PP hypothesis space defined over B-Splines.*

2.2 Learning Rules

We now move beyond simply learning weighted atoms to learning complex dependencies between subspaces in a mixed discrete-continuous setting, that is, probabilistic rules for the hybrid atoms. For this purpose, we leverage first-order relational rule learners. The basic idea is to augment the original dataset that uses continuous variables (such as `height(X)`) together with instances of the invented predicates (such as `height_low(X)`, standing for values in the lower range), as determined by the partitioning.

At this stage, there are multiple choices for rule learning. In the simplest setting, we ignore the learned densities and perform rule learning for one or many target predicate(s). In a more advanced setting, we integrate the rule learner directly into the pipeline, giving it not only the predicate but also the learned probabilities as input. Both schemes are considered in the literature. For example, [13] develop a case where parameters and the structure are optimized jointly. In contrast, [44] apply a deterministic rule-learner in a deterministic setting, and the weights are then obtained in a second

step. In this section, we explore both settings as an illustration of our framework's generality.

Definition 3 (Rule Learning with Hybrid Atoms) *Given:*

1. (a) *a set of examples E, consisting of pairs* (x_i, p_i) *where* x_i *is a ground fact for the invented predicate t and* p_i *is the integrated target probability; or*
   (b) *a set of examples E consisting of* $\{x_1, ..., x_n\}$ *where* x_i *is a ground fact for the invented predicate t.*
2. *A background theory B containing information about the examples in the form of a ProbLog program;*
3. *A loss function* $loss(H, B, E)$*, measuring the loss of a hypothesis (set of clauses) H w.r.t B and E;*
4. *A space of possible clauses* L_h *specified using a declarative bias;*

Find: *A hypothesis* $H \subseteq L_H$ *such that* $H = arg\ min_H\ \ loss(H, B, E)$.

In other words, (1) partitions an attribute into its pieces and adds each piece as an atom. The two cases decide whether the integrated probability is passed to the rule learner (e.g. `0.31 :: height_low(X)`) or not (e.g. `height_low(X)`). Then, (2)-(4) realise standard first-order rule learning with the addition that rules (may) have a probability assigned to them. The loss function is, of course, determined by the rule learner itself. ProbFOIL+'s loss function, for example, is based on the error of predictions:

$$loss(H, B, E) = \sum_{x_i, p_i \in E} |P_s(H \cup B \models x_i) - p_i|$$

To compute probabilities for the first case we need to integrate the pieces over their respective polynomial. For each clause c_i and each piecewise polynomial density $\delta_i(x)$ over cutpoints the integral is calculated as $[c_{i-1}, c_i]$, $i \in \{1, \ldots, l\}$: $p_i = \int_{c_{i-1}}^{c_i} \delta_i(x)dx$ where p_i is a constant that denotes the probability over the interval $[c_{i-1}, c_i]$. The hybrid atom is now transformed into a discrete atom with a probability mass, that is, $p_i :: t_i(X)$. This is then used instead of the polynomial, and analogously, such a transformation is applied to all invented predicates.

Once the weights are computed any discrete FOL rule learner can be used to induce the rules. We utilize ProbFOIL+ [13] and SLIPCOVER [8] for the case of probabilistic facts versus not. It should be clear that our architecture does not hinge on either learner and indeed, ProbFOIL+ also handles the case of non-probabilistic facts. Our choices are to be seen as illustrative of the generality of the framework. We immediately also get the following

Theorem 2 *The algorithmic scheme described above realises both versions of Definition 3.*

Proof. It is immediate to see that by integrating the polynomials, we obtain standard probabilistic atoms of the form $p_i :: c_i$, where p_i is a number and c_i is an atom. It now follows that ProbFOIL+ (and SLIPCOVER respectively) performs rule induction over probabilistic examples (and non-probabilistic examples respectively) wrt to the appropriate loss function. □

As a final step, a WMIProbLog program (discussed in the following section) is obtained that combines the individual relations, their polynomial weights and the rules.

3 WMIProbLog

Learning rich representations is only appealing when it is also possible to efficiently query those representations. In particular, PLPs are particularly useful for computing conditional queries against structured models. Our learning regime culminates in the synthesis of WMIProbLog programs, a dialect of ProbLog and Hybrid ProbLog [21], the latter being a generalisation of ProbLog to continuous distributions in parametric forms. Hybrid ProbLog provides an inference scheme based on interval propagation and dynamic discretisation. WMIProbLog supports a generic (that is, applies to any density function approximated in PP form) and efficient exact inference methodology via WMI, and as will be discussed below, it does not involve any discretisation or rewriting of the program.

Syntactically, ProbLog vs Hybrid ProbLog vs WMIProbLog can be contrasted as follows. Consider a simple mixture model in ProbLog, where a biased coin toss yields either a 50% chance of success or a 25% chance of success:

```
0.7::heads. 0.5::a. 0.25::b.
tails :- \+heads.
mix :- heads, a.
mix :- tails, b.
```

To model the chance of success as continuous random variables in Hybrid ProbLog, we might change the corresponding clauses to:

```
(X,uniform(0,10))::a(X).
(X,uniform(0,20))::b(X).
mix(X) :- heads, a(X).
mix(X) :- tails, b(X).
```

In WMIProbLog, in line with our previous sections, those continuous variables over, say, polynomials of degree 0 (chosen for ease of presentation, although they can be of arbitrary degree in general) would be:

```
0.1::a(X) :- between(0,10,X).
0.05::b(X) :- between(0,20,X).
```

Querying for `mix` in the first, and for `mix_chance`:

```
mix_chance :- mix(X), between(0,5,X).
```

in the other programs yields .425, for example.

The semantics of Hybrid ProbLog and WMIProbLog is given by probability distributions over subsets of the facts and values for the numeric variables in the continuous facts, much like in ProbLog, except that these values may be drawn from an uncountable set. (See [21] for details.) But what is perhaps most interesting about WMIProbLog is that, on the one hand, it is simply a syntactic variant of the Hybrid ProbLog program in which continuous functions are not necessarily parametric, but on the other, by integrating probabilities as shown earlier, it is also a well-defined ProbLog program.

Theorem 3 *Given any WMIProbLog program Δ there is a syntactically well-defined ProbLog program Δ^- such that for all $q, e \in \Delta^-$, marginalization and probabilistic querying have the same values in Δ and Δ^-.*

Proof. As in the previous theorem, we simply integrate the polynomials to obtain classic ProbLog probabilistic facts. Clearly, as the query and evidence atoms only refer to these *abstracted* facts, the values for marginalisation and conditional querying will be the same. □

We write $q, e \in \Delta^-$ to mean that the query and evidence atoms only refer to the *abstract program* Δ^-, where continuous atoms involving intervals and polynomials and replaced with integrated probabilities. This idea, of abstracting continuous features is at the very heart of WMI [6]. However, such a strategy would only allow the most trivial conditional probability computations in general (e.g., probability of `mix` given that `heads`), and would not be able to handle queries like `mix_chance`. Following Hybrid ProbLog, one approach would be to syntactically rewrite the program (or dynamically discretise at computation time), so as to involve every interval mentioned in a query in a well-defined manner; e.g., split the definition of $a(X)$ as:

```
.1::a(X) :- between(0,5,X).
.1::a(X) :- between(5,10,X).
```

Such a strategy is acceptable formally but can be seen to be a painful exercise. So, the next observation to make is that if the weights are dropped, we also immediately have a syntactically well-defined ProbLog (and Prolog) program: Given any WMIProbLog program Δ there is a syntactically well-defined ProbLog program Δ^* that is obtained by dropping all the (polynomial and numeric) weights.

What is the advantage of the reduction? To see this, consider that from a computational perspective, the ProbLog engine proceeds by grounding the input program and queries, breaking loops if necessary, to ultimately yield a propositional formula [18]. That formula is then compiled to a data structure such as a BDD or d-DNNF [10]. It is clearly useful to leverage that pipeline. For example, contrast the BDD (below) obtained for the query `mix(X)`, with the one obtained for a query that uses intervals not explicitly mentioned in the original program:

```
a_half :- a(X), between(0,5,X).
query(a_half).
```

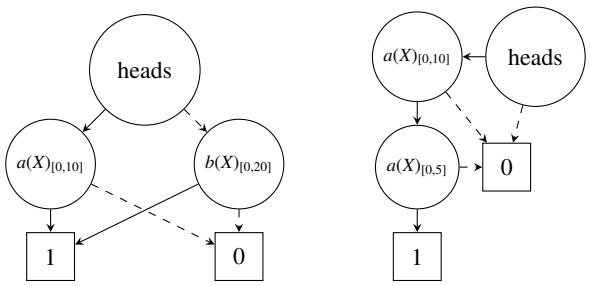

Fig. 1: Proofs of the query in BDD form.

So the only aspect that is needed to be resolved is the computation involving the densities and weights. This becomes possible because ProbLog's computational architecture is immediately generalisable to any commutative semiring [23].[5] (For example, polynomials with natural number coefficients form a commutative semiring.) So the densities can be thought of as abstract elements of a semiring, which can then be symbolically integrated. For example, `query(a_half)` would be computed: $\int_0^{10} .1 \cdot \mathbb{I}_{[0,5]}(x)dx = \int_0^5 .1dx = .5$. Correspondingly, the probability for `mix_chance` can be computed as: $\int_0^{10} .1 \cdot \mathbb{I}_{[0,5]}(x) \cdot .6dx + \int_0^{20} .05 \cdot \mathbb{I}_{[0,5]}(x) \cdot .4 = .425$.

4 Empirical Evaluations

We now evaluate our framework in this section. We reiterate that, *to the best of our knowledge, this is the first attempt to articulate a compositional PLP framework for arbitrarily complex distributions from continuous data.* Thus, our goal was to study the empirical behaviour of the implemented framework on various representational aspects to illustrate its capabilities.

4.1 Datasets

To carefully monitor the quality of the learned models, we mainly utilized the *University data set* [19, 41]. Beyond this, we considered many more hybrid datasets from the UCI repository [15]. The datasets capture domains ranging from health care to marketing, but not all of them are of the same quality. Some, especially *Anneal-U* and *CRX* contain many missing values. Some others contain duplicate entries.

4.2 Learning Representations

On Piecewise Polynomials This section discusses our observations for the piecewise polynomial representation learning. The BIC score determined the model parameters such as the order of polynomials and bins. Table 1 lists statistics for the UCI datasets, which we contextualise further below.

Q1: How does the polynomial learner compare to other density estimators? We compared the polynomial learner to Gaussian mixture models (GMM), which is the current state-of-the-art for density estimation, by appealing to KL divergence in a controlled experiment. (Unimodal Gaussians are used by most PLPs, e.g, [35].) For the GMM learner, we use the one from Python *sklearn*, which also uses the BIC score as a model selection criterion. As polynomial densities are only defined in $\Omega = [cp_0, cp_l]$ the GMMs need to be normalised for a direct comparison. They are multiplied by a normalisation constant $1/\int_\Omega f(x)dx$.

As it is not feasible to calculate the KL divergence directly for GMMs we chose a sample size of 10^6 for a sampling-based approximation. This is calculated as: $KL =$

[5] In an independent and recent effort, Martires et al. [30] have also considered the use of semirings to do WMI inference over propositional circuits.

Dataset	Train	# C	# Bins	O
Anneal-U	673	6 (39)	10.7 (19)	2
Australian	517	5 (15)	7.5 (7.5)	6
Auto	119	15 (26)	6.7 (8.7)	3
Car	294	6 (9)	7 (9.2)	4
Crx	488	6 (16)	5.8 (11.7)	4
Diabetes	576	7 (21)	4.6 (9.1)	4
German	750	3 (9)	6.9 (6)	7
German-org	750	3 (25)	9.1 (6.3)	7
Iris	112	4 (5)	3.75 (3.75)	3

Table 1: Statistics on UCI datsets and the polynomial learning component. It shows (in order) the number of training examples, the number of the continuous attributes (number of all attributes in brackets) the number of created bins separately learned for each attribute and then averaged over the number of features in the dataset (brackets show number when including missing values) and the maximum order learned.

n	T	Expon	Gauss	MoG-2	MoG-5
100	G	0.10409	0.01028	**0.00308**	0.12795
	P	**0.01254**	**0.00300**	0.05699	**0.05701**
1000	G	0.03633	**0.00123**	**0.00335**	0.01550
	P	**0.00860**	0.00228	0.00830	**0.01295**
10000	G	0.02959	**0.00001**	**0.00083**	0.01142
	P	**0.00147**	0.00045	0.00125	**0.00419**

Table 2: Approximation comparison of different distributions in terms of KL divergence. Here, n denotes the sample size and T the model type. Abbreviations: Gaussian Mixture Model estimation (G) and Piecewise Polynomial density estimation (P). We tested distributions following Exponential, Gaussian and Mixture of Gaussian with 2 (MoG-2) and 5 (MoG-5) components.

$\int_{\Omega} p(x) \log \frac{p(x)}{q(x)} dx \approx \frac{1}{N} \sum_{i=1}^{N} \log \frac{p(x_i)}{q(x_i)}$ where $x_1, ..., x_n \sim p(x)$. We compared different types of distributions and sample sizes. The distributions were generated randomly. Not surprisingly, the polynomial learner performs gradually better than the GMMs as the true distribution gets less similar to a mixture of Gaussians (see Table 2). What is perhaps more interesting is how well the PP learner estimates Gaussians. For small sample sizes, the polynomial learner still outperforms GMMs when approximating a unimodal Gaussian. When estimating a more complex distribution such as a mixture of five Gaussian our learner fully outperforms the GMM learner even though the underlying distribution is a mixture of Gaussians. This is mainly because the BIC in combination with polynomial density can (a) handle small sample sizes, and (b) not be forced to determine the appropriate number of components for these samples. This shows that the learner generalises well even with small sample sizes thereby avoiding overfitting.

Q2: Which trends in parameter learning can be observed for . . .

Q2.1: . . . **the order?**

The order of the polynomials in the learned models stayed relatively low in a range from 2 to 5. In fact, only three attributes over all UCI datasets learned an order that was higher than 5 but never higher than 7 (see Table 1). The few outliers occurred when an

attribute contained a high number of missing values. Naturally, low order polynomials are computationally simpler to integrate during inference.

Q2.2: ... **the number of intervals?**

The appropriate number of intervals to approximate a PDF increases with its non-uniformity, the unimodal Gaussian only needed 3 pieces, whereas the 5 Gaussian Mixture needed 6 or more (depending on the sample size). Furthermore, to achieve a close approximation, the cutpoints have to span the entire range of attribute values which is why our algorithm often preferred equal frequency over equal width partitions.

We also observed that the BIC is sensitive to the number of attribute instances. For under 100 data points, the smallest partition and order was often chosen (cf. Table 1). This observation helps to explain the behaviour when missing values are observed. If a dataset contained missing values, we filtered them out in a preprocessing step leading to less samples for the attribute so as to not skew the learner. However, our observations indicate that even though more values achieve a higher score, the smaller representation is still able to generalise well.

4.3 Rule Learning

Here, we report on how the invented predicates integrated with the discrete rule learners.

```
grade_high(A,B) :- sat_low(A,B), \+diff_hard(B), \+iq_1(A,X).
grade_high(A,B) :- takes(A,B), \+diff_hard(B).
grade_high(A,B) :- nrhours_4(B,Y), iq_1(A,X).
grade_high(A,B) :- nrhours_4(B,Y), course(B).
grade_high(A,B) :- sat_low(A,B)
0.056367 -0.001598* X + 1.25739e-05 * X ^ 2 ::
iq_1(A,X):-iq(A,X),between(52.6,103.4,X)
```

Table 3: Example rules learned by ProbFOIL+ (above horizontal line) and SLIPCOVER (below line) for the target `grade_high(A,B)`. Below these a sample predicate is shown, the polynomial is given as a weight before the rule head `iq_1(A,X)` and the body of the rule determines the interval start and end points where the polynomial density holds.

Q3: How do the invented predicates interact with the rule learners? The idea behind these invented predicates is to capture continuous data in terms of PP density approximations, which are ultimately adapted to the input format of the rule learners. Table 3 shows an example. The invented predicates are distinguished by the underscores followed by numbers. In the case of the `iq` predicate, higher number indicate higher values. The grade of a course being influenced by a student's intelligence is clearly intuitive: a student either gets a high grade if their intelligence is not low (rule 1) or if they have low intelligence and put in a lot of hours of study (rule 3), and so, the rules provide a logical way to make sense of dependencies between subspaces in a mixed discrete-continuous settings. We further see in Table 4 that the rule accuracy is not affected over larger domains, which is clearly desirable.

Q4: How do the rule learners compare? As can be seen in Table 4, our setting is not necessarily indicative of any general trend for the two rule learners, but we did

grade	L	AUCPR	AUCROC	Th	Pr
high	S	0.249906564	0.750555262	8	1.375
	P	**0.381490547**	**0.935926625**	2	2.5
mid	S	**0.404951538**	**0.940245155**	7	1
	P	0.3914379	0.936826911	2	2
low	S	0.152929751	**0.813107006**	10	1.4
	P	**0.166327753**	0.654570657	1	3

Table 4: Statistics for the predicates `grade_high, grade_mid, grade_low` for the two rule learners (L): SLIPCOVER (S) and ProbFOIL+ (P). Comparisons are drawn in terms of area under the precision/recall curve (AUCPR), area under the ROC curve (AUCROC), the number of rules in a theory (Th) and the average number of literals per rule for a theory (Pr).

see, for example, that SLIPCOVER rules are shorter than ones learned via ProbFOIL+, on average. In terms of standard measures, we found both to be sufficient for inducing reasonable programs.

Q5: How efficient is the query engine? In order to test the efficiency of our in-

P	t=5	t=10	t=20	t=50	t=100
2	0.9514	1.0506	1.2817	2.1540	4.0185
5	1.2804	1.4325	1.6293	2.5331	4.3767
10	1.4885	1.5773	1.8441	2.7239	4.6914
15	1.8313	1.9621	2.2031	3.1954	5.1372

Table 5: Query execution time in seconds. P denotes the number of pieces of the underlying attributes, and t the amount of terms in the query.

ference engine over large programs, we generated 100 random continuous features and tested the query computation time. We fixed the number of bins to 2, 5, 10 and 15 in order to study the computation time. The query length, e.g. the number of terms in the query was set to 5, 10, 20, 50 and 100. The results are shown in Table 5. As can be seen, the system scales linearly with increasing sizes, which is desirable.

5 Related Work

Inference and learning in probabilistic systems are the most fundamental problems in uncertainty reasoning within AI, to which our work here contributes. There is an important distinction to be made between graphical models (including relational counterparts such as Markov logic networks [42]) and the inductive logic programming machinery that we use here, the latter based on logical deduction and proof theory. A comprehensive discussion on the semantic subtleties would be out of scope, and orthogonal to the main thrust of the paper. We refer interested readers to the lineage discussion in [13, 39], and the possible worlds perspective in [43]. Indeed, our fundamental contributions include the dynamic interval and the polynomial density construction, all of which can

perhaps be easily applied to other formalisms. (Naturally, the choice of the underlying induction paradigm may affect the nature and interpretability of the rules learned.)

From an inference viewpoint, much of the literature is focused on discrete random variables for which approaches such as model counting and bucket elimination have strong runtime bounds [9, 37]. In contrast, WMI serves as a computational abstraction for exact and approximate inference in hybrid domains [6, 5] based on piecewise polynomial approximations. It is thus different in spirit to variational methods and analytic closed-form solutions for parametric families. We refer the reader to [6] for discussions.

On the learning front, several feature and structure induction techniques have been proposed primarily for discrete data (or discretised models of continuous data). Indeed, learning relational features from data is very popular in NLP and related areas [40, 44]. Rule learning has been studied in many forms, e.g., [17, 26].

We leverage ProbFOIL+ and SLIPCOVER in this paper. ProbFOIL+ is based on the mFOIL rule learner [17]. Later work, such as nFOIL [26] integrates FOIL with Bayesian learning, and kFOIL [27] admits the specification of kernel methods with FOIL. Early versions of Markov logic network structure learning algorithms, for example, combined ideas from ILP and stochastic greedy search from classical graphical modelling communities [24]. Approaches such as [36, 2] have further applied rule learning to complex applications such as automated planning and affordances. SLIPCOVER, on the other hand, is based on SLIPCASE [7]. In general, the structure is learned through an iterative search through the clause space.

Treating continuous and hybrid data in such contexts, however, is rare. Existing inference schemes for hybrid data are either approximate, e.g., [33, 1], or make restrictive assumptions about the distribution family (e.g., Gaussian potentials [28]). Structure learning schemes, consequently, inherit these limitations, e.g., [35, 41] where they learn rules by assuming Gaussian base atoms.

6 Conclusions

To the best of our knowledge, this is the first attempt to articulate a learning regime for inducing PLPs from continuous data. It contributes an algorithmic framework that learns piecewise polynomial representations which are then integrated with rule learning to obtain probabilistic logic programs, along with effective symbolic inference. In our view, this work takes a major step towards the difficult challenge of inducing representations from data, especially continuous and mixed discrete-continuous data. In the long term, we hope the declarative/interpretability aspect of our proposal will prove useful when reasoning and learning over uncertain hybrid data.

References

1. Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., Zese, R.: cplint on swish: Probabilistic logical inference with a web browser. Intelligenza Artificiale **11**(1), 47–64 (2017)
2. Antanas, L., Frasconi, P., Costa, F., Tuytelaars, T., De Raedt, L.: A relational kernel-based framework for hierarchical image understanding. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). pp. 171–180. Springer (2012)

3. Baldoni, V., Berline, N., De Loera, J., Köppe, M., Vergne, M.: How to integrate a polynomial over a simplex. Mathematics of Computation **80**(273), 297–325 (2011)
4. Baral, C., Gelfond, M., Rushton, J.N.: Probabilistic reasoning with answer sets. TPLP **9**(1), 57–144 (2009)
5. Belle, V., Van den Broeck, G., Passerini, A.: Hashing-based approximate probabilistic inference in hybrid domains. In: UAI (2015)
6. Belle, V., Passerini, A., Van den Broeck, G.: Probabilistic inference in hybrid domains by weighted model integration. In: Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI). pp. 2770–2776 (2015)
7. Bellodi, E., Riguzzi, F.: Learning the structure of probabilistic logic programs. In: International Conference on Inductive Logic Programming. pp. 61–75. Springer (2011)
8. Bellodi, E., Riguzzi, F.: Structure learning of probabilistic logic programs by searching the clause space. Theory and Practice of Logic Programming **15**(2), 169–212 (2015)
9. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artificial Intelligence **172**(6-7), 772–799 (2008)
10. Darwiche, A., Marquis, P.: A knowledge compilation map. Journal of Artificial Intelligence Research **17**, 229–264 (2002)
11. De Boor, C., De Boor, C., Mathématicien, E.U., De Boor, C., De Boor, C.: A practical guide to splines, vol. 27. Springer-Verlag New York (1978)
12. De Maeyer, D., Renkens, J., Cloots, L., De Raedt, L., Marchal, K.: Phenetic: network-based interpretation of unstructured gene lists in e. coli. Molecular BioSystems **9**(7), 1594–1603 (2013)
13. De Raedt, L., Dries, A., Thon, I., Van den Broeck, G., Verbeke, M.: Inducing probabilistic relational rules from probabilistic examples. In: Proceedings of 24th international joint conference on artificial intelligence (IJCAI). pp. 1835–1842 (2015)
14. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Machine Learning **100**(1), 5–47 (2015)
15. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017), `http://archive.ics.uci.edu/ml`
16. Dougherty, J., Kohavi, R., Sahami, M., et al.: Supervised and unsupervised discretization of continuous features. In: Machine learning: proceedings of the twelfth international conference. vol. 12, pp. 194–202 (1995)
17. Džeroski, S., Cestnik, B., Petrovski, I.: Using the m-estimate in rule induction. Journal of computing and information technology **1**(1), 37–46 (1993)
18. Fierens, D., Van den Broeck, G., Thon, I., Gutmann, B., Raedt, L.D.: Inference in probabilistic logic programs using weighted CNF's. In: UAI. pp. 211–220 (2011)
19. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Relational data mining, pp. 307–335. Springer (2001)
20. Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: A language for generative models. In: Proceedings of UAI. pp. 220–229 (2008)
21. Gutmann, B., Jaeger, M., De Raedt, L.: Extending problog with continuous distributions. In: ILP. vol. 6489, pp. 76–91. Springer (2010)
22. Islam, M.A., Ramakrishnan, C., Ramakrishnan, I.: Parameter learning in prism programs with continuous random variables. arXiv preprint arXiv:1203.4287 (2012)
23. Kimmig, A., Van den Broeck, G., De Raedt, L.: An algebraic prolog for reasoning about possible worlds. In: Proc. AAAI (2011), `http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3685`
24. Kok, S., Domingos, P.: Learning the structure of Markov logic networks. In: Proceedings of the International Conference on Machine Learning. pp. 441–448 (2005)
25. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)

26. Landwehr, N., Kersting, K., De Raedt, L.: nfoil: integrating naïve bayes and foil. In: AAAI-05. pp. 795–800 (2005)
27. Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P., et al.: kfoil: Learning simple relational kernels. In: Aaai. vol. 6, pp. 389–394 (2006)
28. Lauritzen, S.L., Jensen, F.: Stable local computation with conditional gaussian distributions. Statistics and Computing **11**(2), 191–203 (2001)
29. López-Cruz, P.L., Bielza, C., Larrañaga, P.: Learning mixtures of polynomials of multidimensional probability densities from data using b-spline interpolation. International Journal of Approximate Reasoning **55**(4), 989–1010 (2014)
30. Martires, P.Z.D., Dries, A., Raedt, L.D.: Knowledge compilation with continuous random variables and its application in hybrid probabilistic logic programming (2018), `http://arxiv.org/abs/1807.00614`
31. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: Probabilistic models with unknown objects. In: Proc. IJCAI. pp. 1352–1359 (2005)
32. Muggleton, S.: Inverse entailment and progol. New generation computing **13**(3), 245–286 (1995)
33. Murphy, K.P.: A variational approximation for bayesian networks with discrete and continuous latent variables. In: UAI. pp. 457–466 (1999)
34. Nitti, D., Laet, T.D., Raedt, L.D.: A particle filter for hybrid relational domains. In: IROS. pp. 2764–2771 (2013)
35. Nitti, D., Ravkic, I., Davis, J., De Raedt, L.: Learning the structure of dynamic hybrid relational models. In: ECAI 2016. vol. 285, pp. 1283–1290 (2016)
36. Pasula, H., Marthi, B., Milch, B., Russell, S.J., Shpitser, I.: Identity uncertainty and citation matching. In: NIPS. pp. 1401–1408 (2002), `http://papers.nips.cc/paper/2149-identity-uncertainty-and-citation-matching`
37. Poole, D., Bacchus, F., Kisyński, J.: Towards completely lifted search-based probabilistic inference. CoRR **abs/1107.4035** (2011)
38. Quinlan, J.R.: Learning logical definitions from relations. Machine learning **5**(3), 239–266 (1990)
39. Raedt, L.D., Kersting, K., Natarajan, S., Poole, D.: Statistical relational artificial intelligence: Logic, probability, and computation. Synthesis Lectures on Artificial Intelligence and Machine Learning **10**(2), 1–189 (2016)
40. Raghavan, S., Mooney, R.J., Ku, H.: Learning to read between the lines using bayesian logic programs. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1. pp. 349–358. Association for Computational Linguistics (2012)
41. Ravkic, I., Ramon, J., Davis, J.: Learning relational dependency networks in hybrid domains. Machine Learning **100**(2-3), 217–254 (2015)
42. Richardson, M., Domingos, P.: Markov logic networks. Machine learning **62**(1), 107–136 (2006)
43. Russell, S.: Unifying logic and probability. Communications of the ACM **58**(7), 88–97 (2015)
44. Schoenmackers, S., Etzioni, O., Weld, D.S., Davis, J.: Learning first-order horn clauses from web text. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. pp. 1088–1098. Association for Computational Linguistics (2010)
45. Schwarz, G.: Estimating the dimension of a model. The annals of statistics **6**(2), 461–464 (1978)
46. Zong, Z.: Information-Theoretic Methods for Estimating of Complicated Probability Distributions, vol. 207. Elsevier (2006)