

Probabilistic Rule Learning

Luc De Raedt and Ingo Thon

Report CW 580, April 2010



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Probabilistic Rule Learning

Luc De Raedt and Ingo Thon

Report CW580, April 2010

Department of Computer Science, K.U.Leuven

Abstract

Traditionally, rule learners have learned deterministic rules from deterministic data, that is, the rules have been expressed as logical statements and also the examples and their classification have been purely logical. We upgrade rule learning to a probabilistic setting, in which both the examples themselves as well as their classification can be probabilistic. The setting is incorporated in the probabilistic rule learner ProbFOIL, which combines the principles of the relational rule learner FOIL with the probabilistic Prolog, ProbLog. We report on experiments that demonstrate the utility of the approach.

Keywords : probabilistic rule learning, statistical relational learning, probabilistic programming.

CR Subject Classification : I.2.6

1 Introduction

Rule learners are amongst the most popular and easiest to use machine learning systems. They learn logical rules from deterministic examples but do not really take into account uncertainty. On the other hand, the graphical model and statistical relational learning community are able to reason about uncertainty but have not yet contributed many approaches to learning logical rules. This paper wants to alleviate this situation by introducing a novel probabilistic rule learning setting. In this setting, logical rules are learned from probabilistic data in the sense that both the examples themselves and their classifications can be probabilistic.

As a motivating example that we will use throughout this paper, consider the following windsurfing problem. It is inspired by Quinlan’s playtennis example. The difference between playing tennis and going windsurfing is that windsurfing typically needs to be planned ahead of time, say on the previous day. The effect is that the weather conditions at the next day will still be uncertain at the time of deciding whether to go surfing or not. The forecast might state that tomorrow the probability of precipitation (pop) is 20%, the wind will be strong enough with probability 70%, and the sun is expected to shine 60% of the time, which could be represented by the facts:

0.2::pop(\mathbf{t}). 0.7::windok(\mathbf{t}). 0.6::sunshine(\mathbf{t}).

where the \mathbf{t} indicates the identifier for the example. Past experience in this case would consist of such descriptions together with a probability value for the target predicate (e.g., 0.7::surfing(\mathbf{t})), which could indicate, for instance, the percentage of persons in our team that enjoyed the activity, the percentage of time that we enjoyed the surfing, etc. This type of data can be represented using a traditional attribute-value table, where the attributes are all boolean and the values are the probabilities with which the attribute is true.

The probabilistic rule learning problem, introduced in this paper, is now to induce a set of rules that allows one to predict the probability of the example from its description. For instance, for the surfing example, the following rules could be induced:

surfing(X):- not pop(X), windok(X).

and

surfing(X):- not pop(X), sunshine(X).

where the argument X specifies the identifier of the example. The first rule states that if the expected precipitation is low and the wind is ok, the surfing is likely to be good. There is thus a declarative logical reading of these rules, but also a probabilistic one. The lower the precipitation is and the higher the probability of windok and sunshine the higher the probability that the surfing will be enjoyable. Using the description of the example

0.2::pop(\mathbf{t}). 0.7::windok(\mathbf{t}). 0.6::sunshine(\mathbf{t}).

under this hypothesis. Assuming all facts in the description are independent, this reduces to

$$\begin{aligned} P(\text{surfing}(\mathbf{t})) &= P((\neg\text{pop}(\mathbf{t}) \wedge \text{windok}(\mathbf{t})) \vee (\neg\text{pop}(\mathbf{t}) \wedge \text{sunshine}(\mathbf{t}))) \\ &= P((\neg\text{pop}(\mathbf{t}) \wedge \text{windok}(\mathbf{t})) \vee (\neg\text{pop}(\mathbf{t}) \wedge \text{sunshine}(\mathbf{t}) \wedge \neg\text{windok}(\mathbf{t}))) \\ &= 0.8 \times 0.7 + 0.8 \times 0.6 \times 0.3 = 0.704 \end{aligned}$$

where the rewriting is needed to make the two events mutually exclusive.

Observe that although the windsurfing example is a toy example, this type of probabilistic data arises naturally in many application domains, such as robotics,

vision, natural language processing and the life sciences. For instance, in a vision context there might be uncertainty about the identity, features or class of the object just observed; cf. also the experimental section.

2 Problem Specification

We first introduce ProbLog, a probabilistic Prolog [1, 2] that allows one to work with probabilistic facts and background knowledge. A ProbLog program consists of a set of definite clauses D and a set of probabilistic facts $p_i :: c_i$, which are facts c_i labeled with the probability p_i that their ground instances $c_i\theta$ are true. It is also assumed that the probabilities of all ground instances $c_i\theta$ are mutually independent.

Given a finite set of possible substitutions $\{\theta_{j1}, \dots, \theta_{jn}\}$ for each probabilistic fact $p_j :: c_j$, a ProbLog program $T = \{p_1 :: c_1, \dots, p_n :: c_n\} \cup D$ defines a probability distribution

$$P(L | T) = \prod_{c_i\theta_j \in L} p_i \prod_{c_i\theta_j \in L_T \setminus L} (1 - p_i)$$

over ground subprograms $L \subseteq L_T = \{c_1\theta_{11}, \dots, c_1\theta_{1i_1}, \dots, c_n\theta_{n1}, \dots, c_n\theta_{ni_n}\}$. ProbLog is then used to compute the *success probability*

$$P_s(T \models q) = \sum_{\substack{L \subseteq L_T \\ L \cup D \models q}} P(L|T)$$

of a query q in a ProbLog program T , where $P(q|L \cup D) = 1$ if there exists a θ such that $L \cup D \models q\theta$, and $P(q|L \cup D) = 0$ otherwise. In other words, the success probability of query q corresponds to the probability that the query q is *entailed* using the background knowledge together with a randomly sampled set of ground probabilistic facts. An example ProbLog program and query is shown above in the **surfing** example. For more details on ProbLog as well as on its efficient implementation, we refer to [2].

We are now able to formalize the problem of *inductive probabilistic logic programming* or *probabilistic rule learning* as follows:

Definition 1. *Inductive probabilistic logic programming is defined as*

Given:

1. $E = \{(x_i, p_i) | x_i \text{ a ground fact for the unknown target predicate } t; p_i \in [0, 1] \text{ the target probability of } x_i\}$, the set of examples;
2. a background theory B containing information about the examples in the form of a probabilistic ProbLog program;
3. a loss function $loss(H, B, E)$, measuring the loss of a hypothesis H (that is, a set of clauses) w.r.t. B and E ;

Find: The hypothesis $H \subseteq \mathcal{L}_h$ for which:

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

This loss function aims at minimizing the standard error of the predictions. The reason for this choice is, on the one hand, that it is the simplest possible choice and, on the other hand, that well-known concepts and notions from rule learning and classification carry over to the probabilistic case when this loss function is used as we shall show.

There are several interesting observations about this problem setting. First, it generalizes both traditional rule learning and inductive logic programming to a probabilistic setting. The propositional case illustrated in the windsurfing example is an example of probabilistic rule learning. Furthermore, when the background theory contains also relations and possibly clauses defining further predicates we obtain an inductive probabilistic logic programming setting. In both cases, the original setting is obtained by assuming that the background theory is purely logical and having as only values for the examples 1 and 0; 1 corresponding to the positive examples and 0 to the negative ones. This is in line with the theory of probabilistic logic learning [3] and the inductive logic programming setting obtained would be that of learning from entailment because examples are facts that are probabilistically entailed by the theory.

Second, as in traditional symbolic learning the goal is to find a set of logical rules that satisfy certain constraints, while the rules themselves do not possess any parameters. To the best of the authors' knowledge this problem has not been studied before. It is also interesting to position this problem in the context of the literature on uncertainty in artificial intelligence. There one typically makes a distinction between parameter learning and structure learning, the latter being an extension of the former in that also in structure learning the parameters have to be estimated. The probabilistic rule learning problem introduced above is in a sense dual to the parameter estimation problem. Indeed, when estimating parameters, the structure of the model is assumed to be given and fixed, while here the parameters (the probability values) are fixed and the structure, that is, the rules are to be learned. It would of course be possible to also extend the problem setting so that induced rules may contain new predicates defined by probabilistic facts with unknown probability values¹. This type of extension would require both rule learning and parameter estimation. In the present paper, we will not consider this setting any further and focus instead on the pure probabilistic rule learning problem because it is this setting that directly upgrades the well established rule-learning problem.

Thirdly, the probabilistic rule learning setting is also related to the work on mining frequent item sets from uncertain data in that there also the items hold with a particular probability; cf. [4] for the propositional case and [5] for the first order case. However, instead of learning rules, they mine for patterns whose expected frequency is high.

3 Analysis

A key difference between the probabilistic and the deterministic setting is that each example e_i now has a target probability p_i as opposed to a 1/0 value. Furthermore, while in the deterministic case one obtains a 1/0 error, the probabilistic case is more subtle. To clarify this, we use p_i to denote the positive and $n_i = 1 - p_i$ the negative part of the example e_i , while $p_{h,i}$ and $n_{h,i} = 1 - p_{h,i}$ denote the positive and negative prediction w.r.t. the hypothesis h , and introduce the following quantities, illustrated in Figure 1 left:

1. the true positive part $tp_i = \min(p_i, p_{h,i})$,
2. the true negative part $tn_i = \min(n_i, n_{h,i})$,
3. the false positive part $fp_i = \max(0, n_i - tn_i)$, and
4. the false negative part $fn_i = \max(0, p_i - tp_i)$.

If the prediction is perfect, that is, if $p_{h,i} = p_i$, then $n_{h,i} = n_i$, then the true positive and negative parts are maximal and the false positive and negative part

¹ This could be realized by adding to each clause in a hypothesis a new probabilistic predicate and a fact containing this predicate with unknown probability value.

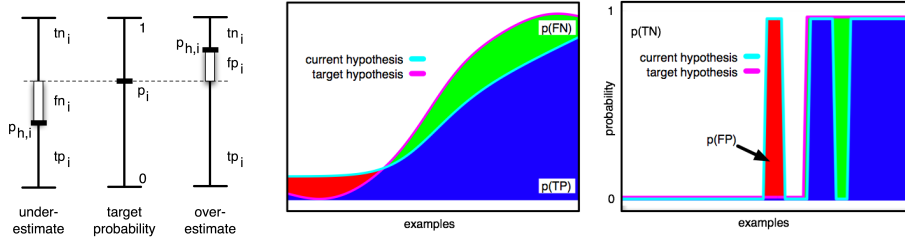


Fig. 1. The true and false positive and negative part of a single example (left) and of an entire dataset for the probabilistic (middle) case, and for the deterministic case (right).

are minimal, that is, 0. However, if $p_{h,i} > p_i$ the hypothesis h overestimates the positive part of the example, and hence, the true positive part is still p_i but the false positive part will be non-zero. Dually, if $p_{h,i} < p_i$, the true negative part is still n_i but the false negative part will be non-zero. Furthermore, let us denote by $TP = \sum_i tp_i$; $TN = \sum_i tn_i$; $FP = \sum_i fp_i$ and $FN = \sum_i fn_i$, that is, the sum of the tp_i, tn_i, fp_i and fn_i , where the sum is taken over all examples in the dataset and by $M = |E|$, $P = \sum_i p_i$ and $N = \sum_i n_i$. These notions could be conveniently shown using a contingency table. It should be clear that this probabilistic contingency table and the above introduces notions directly generalize the deterministic case. To see this, consider that any positive example classified as such will contribute a value of $tp_i = 1$ to TP and $fn_i = 0$ to FN , and any positive example classified as negative will contribute $tp_i = 0$ to TP and $fn_i = 1$ to FN . Using these notions we also define *precision*, *recall* (true positive rate) and *accuracy* using the standard formulas. Thus we have the following property.

The different notions are graphically displayed in Figure 1 (middle,left), in which the x-axis contains the examples and the y-axis their probability and all the examples are ordered according to increasing target probability². The areas then denote the respective rates. The deterministic case is illustrated in the figure 1 (right), which shows that in this case the examples take on 1/0 values. to see that in this deterministic case, the TP, TN, FP and FN correspond to the usual notions of true/false positive/negative rates from the literature in classification, yielding a kind of probabilistic contingency table. Because the TP and FP rates form the basis for ROC space and PN-space, the traditional ROC analysis (as described in, for instance, [6]), used in rule learning can be applied to the probabilistic rule learning setting that we study in this paper and can be interpreted in a similar way as in traditional rule learning. Therefore, ROC analysis techniques, the analysis of heuristics and measures such as AUC essentially carry over to the probabilistic case.

$$\begin{aligned}
 precision &= \frac{TP}{TP + FP} & precision\ m\text{-estimate} &= \frac{TP + m \cdot \frac{P}{N+P}}{TP + FP + m} \\
 recall &= \frac{TP}{TP + FN} \\
 accuracy &= \frac{TP + TN}{TP + TN + FP + FN}
 \end{aligned}$$

² The predicted probability is not necessarily monotone.

Thirdly, the setting could potentially be extended by also allowing that the rules contain new of the rule learning problem and when using a first order background theory (containing also non-unary predicates and possibly clauses defining further predicates) to an ind For the propositional case, one obtains a novel probabilistic rule learning setting. Furthermore, in the first order case, the setting corresponds to a probabilistic extension of inductive logic programming. As in rule learning and inductive logic programming the aim is to find a set of rules, only now the examples and the background knowledge or features of the examples can be probabilistic. When the examples and background theory are deterministic, one obtains the usual learning from entailment setting that is so popular in inductive logic programming.

Finally, the reader may notice that we make the assumption that all probabilistic atoms are independent of one another, also those atoms referring to a single example. This assumption will not hold in general [7] but simplifies the problem. In that regard it is similar in spirit to the naive Bayes assumption and we plan to investigate the consequences of this assumption in future work.

4 ProbFOIL: a probabilistic first order rule learner

We now develop a probabilistic rule learner called ProbFOIL that is able to induce probabilistic logic programs from examples. The rule learner is simple in that it follows the standard and generally accepted principles of rule learners (as described by [8,6]) but does not incorporate advanced pruning strategies because these are unnecessary for showing the relevance of our problem setting and it is straightforward to extend our approach with such techniques. While developing ProbFOIL we started from the generic separate and conquer paradigm (sometimes called the sequential covering algorithm) and modified it as little as possible. Essentially, the algorithm repeatedly adds clauses to the hypothesis in the outer loop until adding further clauses decreases the quality of the hypothesis. Furthermore, while searching for the next clause (lines 5-8) it searches greedily according to some local scoring function in which it repeatedly adds literals to the current clause until some local stopping criterion is satisfied. To determine the possible literals, a refinement operator ρ is applied to the current clause; cf. [3]. We also employ list notation for the bodies of the rules, where the notation $[b, l]$ denotes the result of appending the literal l to the body b . The post-processing step of the rule in lines 9-11 implements a kind of post-pruning akin to that in IREP [9]. The resulting algorithm is very much like the standard rule-learning algorithm known from the literature; cf. [8,6].

While the algorithm is similar to that of typical rule-learners, it is important to realize that there are also some subtleties. First, adding clauses to the hypothesis for the target predicate is a monotonic operation, that is, it can only increase the probability of an individual example because adding a clause results in extra possibilities for proving that the example is true. More formally:

Property 1. For all hypotheses H_1, H_2 : $H_1 \subset H_2 \rightarrow TP(H_1) + FP(H_1) \leq TP(H_2) + FP(H_2)$.

Interpreted in Figure 2, adding clauses to a hypothesis can only increase the red and blue regions, that is, move them upwards, and at the same time reduce the white and green ones. This explains why ProbFOIL stops adding clauses to the hypothesis when adding the rule found last does not result in a better global score. This is akin to the standard stopping criterion employed in many rule learners. As the global scoring function we employ $accuracy(H)$.

Secondly, notice that specializing a clause, that is, adding literals to a clause can only decrease the probability of examples (and hence, decrease the red and

Algorithm 1 The ProbFOIL algorithm

```
1:  $H := \emptyset$ ;  
2:  $h := t(X_1, \dots, X_n)$  where  $t$  is the target predicate and the  $X_i$  distinct variables;  
3: while  $\text{globalscore}(H) > \text{globalscore}(H \cup \{c\})$  do  
4:    $b := []$ ; initially the body of the rule is empty;  
5:   while  $\neg \text{localstop}(H, h \leftarrow b)$  do ▷ Grow rule  
6:      $l := \arg \max_{l \in \rho(h \leftarrow b)} \text{localscore}(h \leftarrow [b, l])$   
7:      $b := [b, l]$   
8:   let  $b = [l_1, \dots, l_n]$   
9:    $i := \arg \max_i \text{localscore}(H, h \leftarrow l_1, \dots, l_i)$ ;  
10:   $c := p(X_1, \dots, X_n) \leftarrow l_1, \dots, l_i$ ;  
11:  if  $\text{globalscore}(H) < \text{globalscore}(H \cup \{c\})$  then  
12:     $H := H \cup \{c\}$   
13: return  $H$ 
```

blue regions).

Property 2. For all hypotheses H and clauses $h \leftarrow l_1, \dots, l_n$ and literals l : $TP(H \cup \{h \leftarrow l_1, \dots, l_n\}) + FP(H \cup \{h \leftarrow l_1, \dots, l_n\}) \leq TP(H \cup \{h \leftarrow l_1, \dots, l_n, l\}) + FP(H \cup \{h \leftarrow l_1, \dots, l_n, l\})$

Thirdly, while traditional deterministic rule learners typically manipulate also the set of examples (e.g. deleting the already covered examples), our probabilistic rule learner takes into account all examples all of the time. In the deterministic case, deleting the already covered examples is warranted because if one rule in the hypothesis covers the example, the overall hypothesis will cover the example. In the probabilistic case, this is more subtle as a given rule may only cover part of the example, and therefore a multi-rule hypothesis may be needed to cover the full positive part of an example. Our algorithm takes this into account in the heuristics used in its inner loop, where it will make decisions based on the extra parts of the examples that become covered by the new rule. In terms of the visualization in Figure 2, this is the difference between the old and new blue and red parts. The local scoring function is based on the m-estimate. This is a variant of precision that is more robust against noise in the training data. However, because each rule may only cover fractions of the examples, we use the difference in m-estimate, i.e.,

$$\text{localscore}(H, c) = \text{m-estimate}(H \cup \{c\}) - \text{m-estimate}(H).$$

Finally, ProbFOIL stops refining rules when the current rule does not cover any extra negative part any more, or when it does not cover any extra positive part any more. More formally,

$$\text{localstop}(H, c) := (TP(H \cup \{c\}) - TP(H) = 0) \vee (FP(H \cup \{c\}) - FP(H) = 0).$$

It should also be clear that standard extensions of rule-learning, such as those for dealing with multiple classes, using beam-search, and look-ahead, can easily be incorporated in ProbFOIL.

5 Experiments

We demonstrate our approach in two experiments. In the first experiment we learned the rules for the **surfing** example, in the second experiments we learned the underlying rules in the *Eulisis* game starting from image data. We used the YAP-ProbLog implementation and computed all scores using exact inference. All experiments were performed on a 3Ghz Machine with 2GB of Ram.

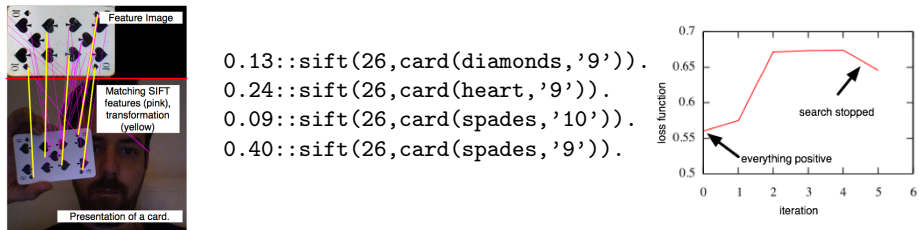


Fig. 2. Sift feature matches for spades nine and spades 10 (left). All matches for the 26th card which is a spades 9 bottom (middle). Accuracy of the ruleset after each iteration on the red implies even dataset (right).

5.1 Surfing

For the surfing example we generated a dataset of 20 training examples starting from the two clauses listed in the introduction. The probabilities of the features were randomly initialized, and we were able to rediscover the original rule set. The runtime was less than 40 seconds. Afterward we tried to rediscover the original ruleset. The ruleset inferred was

1. `surfing(I):-windok(I),\+ pop(I).`
2. `surfing(I):-\+ pop(I),sunshine(I).`
3. `% surfing(I):-sunshine(I),\+ windok(I).`

Please note that when calculating the score of a rule I is ground. This allows to calculate the probability of a negated clause. The last rule decreases the total accuracy therefor it is in accordance to the algorithm not added to the final ruleset and search is stopped. Runtime was less then 40 seconds.

5.2 Eulisis

For the second experiment we used the game of *Eulisis* [10]. *Eulisis* is a game where the dealer has a secret set of rules in mind. For each partial sequence of cards, the rules specify which cards are valid extensions of the sequence and which ones are not. After a card is played, the players are told whether the card is a valid extension or not. Using this information they have to guess the set of secret rules. The concept represented by the rules has to be expressed in terms of the *suit*, *rank*, *color* of the card, and whether it is *even* or a *face* card.

In each players turn he has to extend the current sequence by a card and is told whether this card is positive or negative which is also indicated by the configuration on the table. We played this game against the computer by taking actual images of the played cards (cf. also Figure 2). First, we presented the computer a sequence of 32 cards in a random order. For each card in the sequence the computer is told whether the card is a positive extension of the present partial sequence or not. The cards are classified using *SIFT* (scale-invariant feature transforms) features (corresponding to the start/end points of the pink lines in Fig 2). Each SIFT feature identifies points of interest in the images. The main advantage of *SIFT* features is that they are easy to calculate. While each image contains a large number of features (typically around 1000) normally only a few (~ 70) will match with a prototype (pink lines in Fig. 2). On the other hand, if a consistent transformation (scaling/rotation/translation) of only a small set of features ($\sim 10 - 20$ yellow lines in Fig. 2) between the image and the prototype can be calculated, the probability of a miss-classification is extremely low. To calculate the transformation we used the *RANSAC* (random consensus) algorithm, which automatically eliminates false matches (like in the hair

region in Figure 2). To identify a certain card we took an example picture of this card. When a card was presented to the computer the computer calculates the *SIFT* features of the current scene and the card. Afterward a transformation of the matching features between both images is calculated. False matches are eliminated using the *RANSAC* (random consensus) algorithm. The number of matched features is considered to be proportional to the probability of the card being a match.

The output generated by the image analysis is highly interesting in our context because it often results in confusions between similar cards. Typical confusions occur between cards of the same suit (e.g, the 9 versus the 10) as well as between numbered cards belonging to different suits (e.g., the 9 of hearts versus the 9 of diamonds). These confusions show that this is a realistic setting for probabilistic rule learning.

We used two concepts to test the algorithm. The first sequence contains the concept states that the next card has to be red, that is,

```
trans(PrevPos,Curr):-red(Curr).
```

Learning this concept took 45 seconds. ProbFOIL found this rule but also discovered an extra rule

```
trans(PrevPos,Curr):-black(PrevPos).
```

The last rule is valid but only an artifact as it does not cover any example neither positive nor negative. It disappears, when m of the m -estimates is set to zero.

The second concept to learn was that black cards have to be followed by odd cards and red cards by even cards. The correct ruleset consists therefore of the two rules:

```
trans(PrevPos,Curr):-black(PrevPos),odd(Curr)
trans(PrevPos,Curr):-red(PrevPos),even(Curr)
```

Again, ProbFOIL learned some extra rules covering some very small noisy fractions of examples:

```
trans(PrevPos,Curr):-odd(Curr),even(Curr),red(Curr)
trans(PrevPos,Curr):-black(Curr),even(Curr),odd(Curr).
```

The last two rules are logically inconsistent as cards cannot be even and odd at the same time, but due to the uncertainty of the observations, they are probabilistically possible. In any case it is interesting to see how the accuracy evolves as more rules are learned. This is graphically depicted in Figure 2 right. The accuracy of the rule stating that everything is positive is 0.56, the accuracy of the different rule-sets learned by ProbFOIL are in order of discovery 0.574, 0.671, 0.673, 0.673. This also implies that the target concept itself has only an accuracy of 0.671, and that the last two rules that are added only account for 0.2% accuracy. Thus the improvement of these last two rules is marginal and they would typically be removed should we employ a kind of post-pruning. The improvement after finding the first two rules is only marginal. Post pruning using a separate hold set would probably remove those.

6 Conclusions

We have introduced a novel setting for probabilistic rule learning and developed the ProbFOIL algorithm for solving it using the probabilistic logic programming system ProbLog. The result is a natural probabilistic extension of inductive logic programming and rule learning. Future work will be concerned with the development of efficient propositional and first order rule learners.

References

1. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic Prolog and its application in link discovery. In Veloso, M., ed.: Proceedings of the 20th International Joint Conference on Artificial Intelligence. (2007) 2462–2467
2. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, V., De Raedt, L.: On the efficient execution of problog programs. In: ICLP. (2008) 175–189
3. De Raedt, L.: Logical and Relational Learning. Springer (2008)
4. Chui, C.K., Kao, B., Hung, E.: Mining frequent itemsets from uncertain data. In Zhou, Z.H., Li, H., Yang, Q., eds.: PAKDD. Volume 4426 of Lecture Notes in Computer Science., Springer (2007) 47–58
5. Kimmig, A., Raedt, L.D.: Local query mining in a probabilistic prolog. In Boutilier, C., ed.: IJCAI. (2009) 1095–1100
6. Fürnkranz, J., Flach, P.A.: Roc ‘n’ rule learning—towards a better understanding of covering algorithms. *Mach. Learn.* **58**(1) (2005) 39–77
7. Jäger, M.: [personal communication]
8. Mitchell, T.M.: *Machine Learning*. McGraw-Hill (1997)
9. Fürnkranz, J., Widmer, G.: Incremental reduced error pruning. In Cohen, W., Hirsh, H., eds.: *ML94, MK* (1994) 70–77
10. Dietterich, T., Michalski, R.: Learning to predict sequences. In Michalski, R., Carbonell, J., Mitchell, T., eds.: *Machine learning: An artificial intelligence approach*. Volume 2. MK (1986)