Abnormal Predicates: Learning Categorical Defaults from Probabilistic Rules

Rose Azad Khan and Vaishak Belle
School of Informatics, University of Edinburgh, Edinburgh, U.K.

Keywords: Logic and Learning, Defaults, Knowledge Representation.

Abstract: Learning defaults is a longstanding goal in the field of knowledge representation and reasoning. We provide a novel method for learning defaults by way of introducing a new predicate: the abnormal predicate, which explicitly covers all the exceptions to a rule, thus forming a default theory. Our proposed method for learning defaults is sound and complete for all rule-exceptions, and can be extended for use on other frameworks.

1 INTRODUCTION

The use of default rules is a fundamental concern in knowledge representation and reasoning. To date, a variety of approaches have been proposed for capturing defaults (Lakemeyer and Levesque, 2006), best represented by the example that birds typically fly. What we want as exceptions are certain categories of birds, such as penguins and ostriches, which obviously do not fly. Perhaps the most established account for dealing with defaults is by the use of nonmonotonic logic (Halpern, 1997; Denecker et al., 2000). The idea is that the property of flying could hold for all birds. But for birds that are ostriches, dead, or penguins, we would retract the claim that they fly. That is, the addition of knowledge forces us to retract, leading to the nonmonotonic flavor.

In fact, early in the history of knowledge representation, John McCarthy proposed the notion of circumscription (McCarthy, 1980; Reiter, 1982). Here the idea is that so-called *abnormal predicates* can be used. The idea is that if one carefully controls the extension of this predicate, then the property of flying could hold for all but the unusual ones. Ostriches and dead birds, for example, would be considered abnormal, allowing the property of flying to be inferred for the remaining birds. For ones in such categories, we would conclude that these birds would not fly. In a first-order setting, if a particular bird, such as Tweety, is not abnormal, then we would conclude that Tweety can fly.

The exciting progress made on unifying different semantic approaches for defaults is noteworthy (Etherington, 1987; Boutilier, 1994; Lakemeyer and Levesque, 2006). However, the challenge of automatically learning defaults is still a major one.

We provide a novel method for learning defaults by way of introducing an invented predicate – the abnormal predicate – which takes its inspiration from John McCarthy's proposal. This invented predicate explicitly covers all the exceptions to a rule, thus forming a default theory. Default theories present a way of formalising inference rules without having to explicitly account for all of their exceptions (Etherington, 1987). It has been argued that default theories more closely represent human reasoning and knowledge (Morgenstern and McIlraith, 2011).

Our proposed method for learning defaults is sound and complete for all rule-exceptions, and can be extended for use on other frameworks. In this paper, we showcase a method for learning defaults implemented by way of extending the probabilistic rule-learning algorithm ProbFOIL (De Raedt et al., 2015). However, we treat the probabilistic rule-learner as a black box, and our approach is therefore algorithm-agnostic, and could be implemented by any program which meets the input specifications of our algorithm.

2 BACKGROUND: ProbFOIL

The ProbFOIL algorithm (De Raedt et al., 2015) is a probabilistic extension of the FOIL rule learner (Quinlan and Cameron-Jones, 1993), built using ProbLog syntax (De Raedt et al., 2007). The method of learning rules is similar to standard Inductive Logic Programming (ILP) (Muggleton et al., 2012). An

Khan, R. A. and Belle, V.
Abnormal Predicates: Learning Categorical Defaults from Probabilistic Rules.
DOI: 10.5220/0013229000003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 892-900
ISBN: 978-989-758-737-5; ISSN: 2184-433X

ILP algorithm creates rules by constructing clauses from a set of declared predicates. The hypothesized clauses are evaluated by looking at the number of examples which they entail correctly. The optimal rule set is the one which entails all the positive examples in the dataset and none of the negative examples. Usually, the hypothesis is expected to be consistent with all the examples, and this means that a single incorrect example can prevent an entire rule from being learned. In ProbLog and ProbFOIL, this is not a problem since examples and rules do not have to be categorical. The examples used in ProbFOIL can (optionally) have probabilities attached, and the resulting clauses learned are also probabilistic. When the probabilities of the examples are all set to 0 and 1, the rule-learning problem is the same as standard ILP (De Raedt et al., 2015).

The ProbFOIL algorithm evaluates candidate clauses by considering the number of examples correctly predicted by each clause. An example is an atom which assigns some property (i.e., predicate) to a constant e.g. $bird(a)$. A (non-ground) clause is a disjunction; for example $bird(x) \rightarrow flies(x)$. The aim is to find the clause (or set of clauses) which best predicts the examples given.

Predicted examples fall into four categories: true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). True positive examples are those which are positive, and are correctly predicted as positive. False positives are incorrectly predicted as positive. True negatives are correctly predicted negative examples, and false negatives are incorrectly predicted as negative. The ProbFOIL algorithm tries to find rules which maximize the number of true positives, while minimizing the number of false positives and false negatives. Since this is a probabilistic setting, both the classification of an example and the predictions of a hypothesis are probabilistic i.e. a given hypothesis will predict a probability value $p_i \in [0,1]$ for an example e_i, instead of 0 or 1.

As discussed, in a deterministic setting, $p_i = 1$ for a positive example, and $p_i = 0$ for a negative example. This corresponds to a standard ILP model. In the probabilistic setting, each example e_i adds p_i to the positive part of the dataset, and $1 - p_i$ to the negative:

$$P = \sum_{i=0}^{M} p_i$$

$$N = \sum_{i=0}^{M} (1 - p_i) = M - P$$

P denotes the positive part of the dataset, and N the negative part. M is the size of the dataset i.e. the total number of examples in the learning data.

The true positive and false positive rates of the model (TP and FP resp.) are therefore not integers, but are the sums of the probabilities of the examples:

$$TP_H = \sum_{i=0}^{M} tp_{H,i} \text{ where } tp_{H,i} = min(p_i,\ p_{H,i})$$

$$FP_H = \sum_{i=0}^{M} fp_{H,i} \text{ where } fp_{H,i} = max(0,\ p_{H,i} - p_i)$$

TP_H and FP_H refer to the true positive and false positive rates of the model under hypothesis H. Here, p_i is the true probability of example e_i, and $p_{H,i}$ is the predicted probability of the example under hypothesis H. Furthermore, $TN_H = N - FP_H$ and $FN_H = P - TP_H$. This is also the case in the deterministic setting (De Raedt et al., 2015).

The algorithm operates using two loops. The outer loop maximizes a global scoring function, while the inner loop maximizes a local scoring function. The global scoring function used is accuracy, which is a measure of the proportion of examples correctly classified by the algorithm:

$$accuracy_H = \frac{TP_H + TN_H}{M}$$

The local scoring function uses the m-estimate, which is a variant of precision. Precision measures the proportion of examples classified as true which are actually true. The m-estimate is given by:

$$m - estimate_H = \frac{TP_H + M\frac{P}{N+P}}{TP_H + FP_H + m}$$

In the above equation, m is a parameter of the algorithm. P is the total number of positive examples, and N is the total number of negative examples. TP and TN are the same as above, while FP is the number of false positives.

The algorithm operates as follows: the outer loop begins with an empty set of clauses, and repeatedly adds clauses to the hypothesis until no more improvement of the m-estimate can be obtained. The inner loop obtains the clauses to be added to the outer loop. Given a set of clauses H, the algorithm searches for the clause $c(x) = (x :: c)$ the clause which maximizes the local scoring function. Here, x indicates the probability that the body of the clause entails the head. The algorithm attempts to maximise x according to:

$$arg\ max_x\ M(x) = \frac{TP_{H \cup C(x)} + M\frac{P}{N+P}}{TP_{H \cup C(x)} + FP_{H \cup C(x)} + m}$$

For each candidate clause $c(x)$, examples are classified according to whether their target value p_i is overestimated, underestimated or estimated perfectly by

the clause. If it is the case that $p_{H,i} > p_i$ then the true positive part is p_i, and the remaining value $p_{H,i} - p_i$ belongs to the false positive part of the dataset. In this case, hypothesis H overestimates the value of e_i. If the hypothesis underestimates e_i such that $p_i > p_{H,i}$ then $p_{H,i}$ is the true positive part, and $p_i - p_{H,i}$ belongs to the false negatives. From these categories of examples, the true positive and false positive rates are calculated, and hence the m-estimate is calculated.

The ProbFOIL learning algorithm uses a beam search in order to find the global rather than local maximum. Clauses are generated using relational path finding, which is a method of considering connections between variables in the examples (De Raedt et al., 2015). The probabilities $p_{H,i}$ are computed using functionality from ProbLog (De Raedt et al., 2007). The algorithm also includes a significance test in order to penalise long clauses. Often, long clauses will have a number of literals in the body which do not contribute much to the overall predictive power of the clause. The significance test used here is a variant of the likelihood ratio statistic (De Raedt et al., 2015). The significance level p can be set on the command line. For this project, we used $p = 0.0$ since our implementation required that long clauses were returned by the algorithm.

3 METHODOLOGY

The aim of this paper is to build a system which first learns probabilistic rules from probabilistic data, using regular, unmodified structure learning methods, and then attempts to 'complete' these statistical rules by constructing an abnormal predicate. As part of this process, the program constructs the domain of the abnormal predicate by finding the examples and categories which the predicate applies to. In the first instance for example, we learn a rule such as $Bird(x) \rightarrow Flies(x)$ which has a probability attached e.g. 0.96. From this rule, we construct a new rule $Bird(x) \wedge \neg Abnormal(x) \rightarrow Flies(x)$ which has a probability of one. This is our new categorical default. The predicate *Abnormal* applies to every instance for which $Bird(x) \wedge \neg Flies(x)$ is true. We also add to the knowledge base a rule which explicitly states which kinds of objects are abnormal, for example $Penguin(x) \rightarrow Abnormal(x)$, since penguins do not fly.

The final default-learning algorithm therefore consists of two distinct stages. In the first stage, the normal ProbFOIL algorithm is run, and returns a set of rules which explicitly show the negative examples of the rules. In the second stage, the modified default-learning algorithm is run, and returns a rule which subsumes all negative examples under a single abnormal predicate.

Our chosen approach allows us to implement categorical default learning as a single step within the ProbFOIL algorithm. It also allows us to compare learning from the unmodified dataset with learning from the dataset with the new abnormal predicate. This comparison is useful as it enables us to make links with the literature on statistical predicate invention, and compare our results with hypotheses and theories from this literature.

3.1 Learning Defaults

The ProbFOIL algorithm uses a significance threshold to penalize rule length and prevent the algorithm from returning rules which have a high number of predicates in the body of the rule. Setting the significance threshold to zero allows the algorithm to return rules where every negative predicate is included in the body of the rule, for example $Bird(x) \wedge \neg Penguin(x) \rightarrow Flies(x)$. This means that after the original learning algorithm has run with a significance threshold of p=0.0, a rule is returned which explicitly covers every negative instance of that rule. Given this rule, we simply extracted the negated predicates in the body of the rule, and constructed the domain of the new abnormal predicate using these categories.

The standard input to ProbFOIL consists of two files: a database file and a settings file. The settings file specifies the mode and type of each predicate, and the database file contains the instances. The function for the default-learning algorithm takes three arguments: a set of ProbFOIL rules, a data object containing the examples (i.e. the data points from the files), and the data files themselves.

3.2 Implementation

To implement the algorithm, we created a new file 'defaults.py' which contained a function 'construct_ab_pred'. This function was called on the hypothesis after the first learning stage had taken place in the main module execution file 'probfoil.py'. The function takes the learned rules, examples and datafiles, and constructs an abnormal predicate from the learned rule. It then writes new instances of the abnormal predicate to the datafile. After the function has been called in probfoil.py, the data files are reloaded, and the learning algorithm is run again over the new data. This time, the rule returned is the categorical default.

Firstly, the ProbFOIL rules are converted to

clauses. A clause consists of a head and body. In the rule $flies(x)$:- $robin(x)$, the predicate $flies(x)$ is the head of the clause, and the predicate $robin(x)$ is the body. The algorithm checks whether the body of the clause is merely 'true' or 'false' and if either is found, the clause is skipped. This is because a rule such as $flies(x)$:- $true$ merely states that everything flies, while $flies(x)$:- $false$ states the opposite. These rules are already categorical rules of a sort, and therefore have no exceptions and no examples that could be placed under an abnormal predicate. Next, if the body of a clause is a conjunction, then the body is turned into a list of predicates. For each predicate in the list, if the predicate is positive then it is added to a list of positive predicates, and if it is negative then it is added to a separate list. For each predicate in the negative predicate list, we retrieve the examples from the data object in which they are stored. The positive predicate is used to construct the abnormal predicate, since it is this predicate which gives the overall category which the rule pertains to. For each positive predicate, we check whether the negative examples are a subset of the examples which the positive predicate applies to. If they are not a subset, then the predicate is dismissed as an option, since it is the wrong category, that is to say, the negative examples cannot be exceptions to this category.

The abnormal predicate name is constructed by appending 'ab_' to the name of the chosen positive predicate. This yields abnormal predicates such as 'ab_bird' and 'ab_person'. To construct the domain of the predicate, we create new data points from the negative examples using the newly created abnormal predicate. For example, if the original data file contained the data points 'penguin(1)' and 'penguin(4)', the examples 1 and 4 would be returned. The new data points created would then be 'ab_bird(1)' and 'ab_bird(4)'. These are written to the database file. We also create mode and type statements for the abnormal predicate, and these are written to the settings file. The original clause and its abnormal predicate are returned as command line output.

3.3 Alternative Approaches

Given that we have a rule for which we want to create an abnormal predicate, the problem that we face is that of trying to find the classes of objects which fall under this predicate. An alternative method of achieving this would be to construct new rules which specify the domain of the predicate (such as the Penguin rule above), and then see whether the categorical default $Bird(x) \wedge \neg Abnormal(x) \rightarrow Flies(x)$ is entailed by the knowledge base when the new rule is included. If the default is not entailed i.e. does not have a probability of 1, then we must go back and change the domain of the predicate, and try again. The problem with this approach is that it requires running the learning algorithm multiple times, for every possible construction of the abnormal predicate, until the correct domain is found for the abnormal predicate, and the new rules and existing knowledge base jointly entail the categorical default. This method would be computationally expensive, and time taken would increase with every type of exception to the original statistical default.

4 EVALUATION

Both qualitative and quantitative evaluations were carried out. Quantitative evaluations focused on comparing the learning speed of the default-learning algorithm with the learning speed of the original ProbFOIL algorithm, for a range of different sized datasets. The qualitative side of the evaluation considered a range of examples, including complex examples in which multiple defaults were learned. There are open questions concerning whether the defaults returned were what we expected, and how we could judge what constitutes correctness in terms of default learning.

4.1 Quantitative Evaluation

For the quantitative evaluation, we investigated learning speed. We generated datasets of increasing size by hand, based on the birds-penguin default. 10% of the data points generated were randomly selected from a set of non-flying birds: [penguin, ostrich, dodo, kiwi]. A further 10% were randomly selected from a set of non-birds: [cat, dog, rabbit]. The remaining 80% of the data points were generated by selecting randomly from a set of flying birds:[robin, blackbird, thrush, eagle, sparrow]. The data points, including bird or non-bird, species of bird, and whether the objects flies or not, were written to an output datafile. The modes and types of each kind of object were written to the output settings file. Once the files were created and filled, they were suitable to be used straightaway as input to the ProbFOIL default-learning algorithm.

4.2 Learning Speed

The default learning algorithm is implemented in two stages. The original learning algorithm runs during the first stage, on unmodified data. After the first stage, the learning data is modified based on the rules

returned, and the same learning algorithm runs on the newly annotated data. The ProbFOIL code has been modified to return the times taken for the first and second stage, as well as the overall time. Therefore, when comparing unmodified learning with default learning, we can simply compare the times for the first stage with the time for the second stage.

Figure 1 shows the time taken to learn a single default in datasets of increasing size, from 10 data points to 100 data points. From the graph, it is clear that the default learning stage is faster than the non-default learning stage. This is also the case as the datasets become much larger. Figure 2 shows the learning speed for datasets of size 100 to size 1000, increasing by 100 data points at each stage.

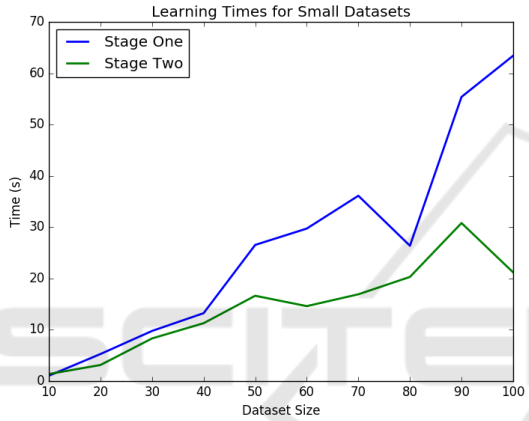

Figure 1.

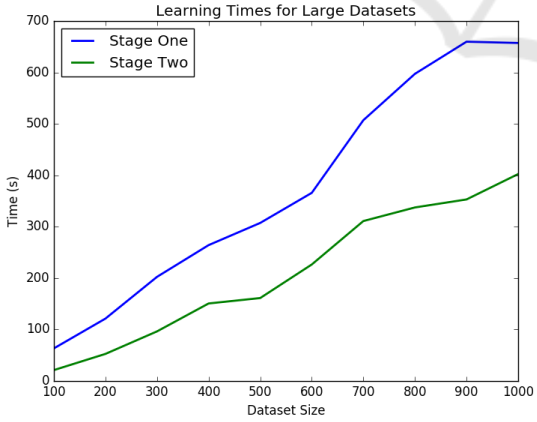

Figure 2.

4.3 Qualitative Evaluation

For the base case, we used a version of the birds example described in the Introduction (Section 1.1). The data consisted of several birds, each of which had a type, and for each bird it was specified whether or not that bird could fly. The types of birds included were: robin, thrush, eagle, blackbird, penguin, dodo and ostrich. We also included dogs, so that birds would be distinguished from other types of objects in the universe. Four of the bird types (thrush, eagle, robin, blackbird) could fly, while the remaining three bird types (penguin, dodo, ostrich) cannot fly. It was also specified that dogs cannot fly. In the learning data, we included several flightless birds, alongside a larger number of flying birds. From the unmodified structure learning algorithm, with a significance level of 0.0, the following rule was returned:

```
flies(x) :-
   bird(x), \+penguin(x),
   \+ostrich(x), \+dodo(x).
```

In our modified version of ProbFOIL, the default-learning stage occurs automatically immediately after the normal structure learning takes place. The algorithm returns the intermediate theory, which is the rule learned from the unmodified data, and the final theory, which is the rule (or rules) learned from the data with the added abnormal predicate data points. The final rule returned in the birds examples is show below:

```
flies(x) :- bird(x), \+ab_bird(x).
```

This is the desired outcome, and the outcome which was to be expected given the data.

We also tested the defaults-learning algorithm on a variety of more complex, interesting examples, including multiple possible defaults in a single dataset. We constructed several examples by hand to investigate.

4.3.1 Independent Defaults

The ProbFOIL algorithm works by specifying a single target predicate at a time, meaning that we cannot test the case of learning multiple defaults simultaneously. Instead we created a dataset which contained two distinct abnormal predicates to learn, and learned first one categorical default, and then the second. The first default learned was the basic birds example. The second rule learned was based on data about several different kinds of cats. The target predicate was ‘tail’. Every kind of cat had a tail, except for injured cats and Manx cats. The rule returned by the normal structure learning algorithm was:

```
tail(x) :-
    cat(x), \+injured(x), \+manx(x).
```

The rule returned by the second stage of the learning process was the categorical default containing the abnormal predicate, as expected:

```
tail(x) :- cat(x), \+ab_cat(x).
```

In this example we first learned the 'flies' predicate, and second learned the 'tail' predicate. The categorical defaults for both predicates were learned successfully.In both cases, the time taken to learn the rule was shorter in the second learning stage (i.e. in the stage in which the ab_pred data points were present) than in the first stage.

4.3.2 Nested Defaults

We use the term 'nested defaults' to mean examples in which first one default is learned, and an abnormal predicate is created, and then a second default which includes the previously learned abnormal predicate is learned. We wanted to see whether the algorithm would make use of the new predicate in the second learning step, and if so whether this affected the outcome of the learning process. The data for this example was based on the usual birds default. We also included data about the origins of the birds (this information was entirely hypothetical and may not reflect the true origins of these bird species). To begin with, we learned the initial rule $flies(x)$:- $bird(x)$, $\backslash + ab\_bird(x)$ in which the abnormal predicate applied to the non-flying birds penguins, dodos, ostriches and kiwis. In the data, we had also specified that dodos, kiwis and ostriches come from Australia, while robins, thrushes, eagles, sparrows and blackbirds are found in England. Penguins are neither English nor Australian. These predicates are mutually exclusive, that is, a bird species cannot be both English and Australian. For completeness, we stated that neither predicate applied to dogs.

For the second learning step, given that we had already learned the abnormal predicate and added the abnormal data points to the data files, we then learned the predicate 'australian(x)'. We compared the learning speeds and outputs for both the normal ProbFOIL algorithm and for the modified default learning algorithm. The output of the normal ProbFOIL algorithm was:

```
australian(x) :-
    \+eagle(x), \+robin(x), bird(x),
    \+penguin(x), \+blackbird,
    \+thrush(x),
```

The time taken for this learning stage was around 9 seconds (varying by 1-2 seconds on each run) and the number of rules evaluated was 220.

The defaults learning algorithm output the following rule:

```
australian(x) :-
    ab_bird(x), \+penguin(x).
```

The same rule was returned at both the intermediate stage and the final stage of the default learning algorithm. On one trial, the time taken for stage one was 13.3 seconds, and the time for stage two was 6.8 seconds, giving a total of 20.1 seconds. There were 99 rules evaluated in stage one and 108 evaluated in stage two. It is interesting to note that in this learning step, the algorithm created an 'ab_ab_bird(x)' predicate, which presumably applied to penguins only, but this predicate was not used in the final rule.

For a second experiment, we learned the predicate 'english(x)'. This predicate applied to all non-abnormal (i.e. flying) birds. When we learned this predicate using the normal algorithm, the following rule was returned:

```
english(x) :-
    bird(x), \+penguin(x), \+dodo(x),
    \+ostrich(x), \+kiwi(x).
```

The learning time was 5.95 seconds and the number of rules evaluated was 189. In the comparison case, we once again learned the typical abnormal predicate *ab_bird*, and then learned the predicate 'english(x)' from the modified data containing the abnormal predicate data points. The following rule was returned:

```
english(x) :- \+ab_bird(x), bird(x).
```

The time taken for stage one was 2 seconds, with 99 evaluations, and for stage two was 1.6 seconds, and once again 99 evaluations. This is remarkably quick compared to the normal algorithm results. However, this speed is perhaps not surprising, since the domain of the english predicate in this example was exactly the same as the domain of the abnormal predicate. This possibly allowed for the learning algorithm to take larger steps, as is described with multi-relational clustering (Kok and Domingos, 2007). Nevertheless, it is clear that this learning stage was quicker due to the abnormal predicate, and that the abnormal predicate data was efficiently used in learning.

In both of the above examples, we can see that a learned abnormal predicate can be used to construct further rules from the same dataset, and that in some cases speeds up the learning process. The increase in learning speed seems to depend on the similarity of the domains which are being learned. This observation supports the results found in statistical predicate invention which state that different clusterings are better at predicting different subsets of the data (Kok and Domingos, 2007). Furthermore, in both cases the rules learned by the default-learning algorithm are notably shorter than the rules learned by the unmodified algorithm.

4.3.3 Conflicting Defaults

In some cases, there may be more than one candidate for the abnormal predicate. We include a loop in the code which tests whether the candidate abnormal predicate (prior to abnormalization) applies to every negative example in the data. If the predicate does not apply to a superset of the set of negative example objects, then it is dismissed as a candidate, since it is clear that the negative examples cannot then be said to be exceptions to this predicate. This approach works when there is more than one positive predicate in the body of the rule, but where only one predicate applies to all the negative examples.

However, this approach will not work when there are two positive predicates which apply to every negative example. For example, assume that we have the same dataset as the original bird example, except that every bird is also specified as being alive. There are no dead birds in this dataset. In this case, there is no functional difference between the 'bird' predicate and the 'alive' predicate, and we have no means of deciding based on the data we have, which category the negative examples are exceptions to. According to the data, it is just as plausible to create an 'ab_alive' predicate to group the negative examples as it is to create an 'ab_bird' predicate.

We have no straightforward answer to this question. It would of course be possible to modify the code to make it possible to create two abnormal predicates for a single rule. The problem we have is that in some cases, it would be incorrect to create one of the predicates, such as in the *ab_alive* case specified above. However, this incorrectness comes from our intuition and our knowledge of the domain, rather than from the data itself. Perhaps in these cases, where it is impossible to distinguish between two candidates for the abnormal predicate, we should either skip this rule, or create two abnormal predicates, under the assumption that at least one is correct. These are open questions, and there is no right answer that we can see. Given our current system, the most we can hope for is that the learning data is complete, and that every possible predicate assignment is present in the data. For example, in the 'alive' case, we would need to include at least one bird that is dead, alongside all the other alive birds, otherwise we might as well just get rid of the 'alive' predicate. In summary, we would like to eliminate the possibility of two predicate domains overlapping completely.

4.4 Limitations

There are several limitations to the default-learning algorithm described here. Firstly, this approach requires that the negative examples are explicitly returned. Secondly, throughout the design and implementation process, we worked on the assumption that the data would be complete and correct. This is of course an assumption that is often not the case when working with real-world datasets.

A final limitation derived from the ProbFOIL algorithm design, which allows only a single target predicate to be specified for learning at a time, meaning that categorical defaults must be learned one-by-one. This is not a problem for small datasets, but may become problematic when we consider large datasets containing multiple possible defaults.

5 RELATED WORK

Surprisingly, there is very little work on learning defaults despite a wide range of languages and semantic approaches for reasoning about defaults (Lakemeyer and Levesque, 2006; Halpern, 1997). There is very early work that discusses learning considerations for representing defaults (Grosof, 1992), but this remains at the level of a conceptual proposal about how defaults can be integrated as inductive bias. Similar in spirit, the work of (Schuurmans and Greiner, 1994) looks at default concepts in terms of a so-called blocking process. By means of a formal justification using Valiant's probably approximately correct learning (Valiant, 1999; Valiant, 2013), the conceptual idea is interesting and possibly applicable to a proposal like ours. However, there is not much on the types of default theories that have been learned beyond some very simple examples.

On the one hand, what we are proposing here is also a conceptual idea: lumping together all the exceptions in one or more abnormal predicates, and appealing to ProbFOIL. In principle, structure learners that are different from ProbFOIL could potentially be leveraged for an implementation of our idea. We have done preliminary work on attempting to recreate this procedure with a Markov Logic Network structure learner (Kok and Domingos, 2010). Over a small dataset that we used for our evaluations, the structure learner returns the following clauses.

```
//predicate declarations
Penguin(dom1).  Flies(dom1).
Bird(dom1).  Robin(dom1).
//rules returned
5.04534 Bird(a1)
-5.76512 Penguin(a1)
```

```
-5.99439 Flies(a1)
-0.00053092 Robin(a1)
10.0914 Bird(a1) v !Robin(a1)
11.6038 !Bird(a1) v Flies(a1)
13.5153 Bird(a1) v !Penguin(a1)
-11.2906 !Penguin(a1) v Robin(a1) v Flies(a1)
```

Here, 'dom1' refers to the domain, and '!' in is equivalent to the negation symbol in FOL.

For instance, it learns that for the constant a1, if it is a robin then it is also a bird. Likewise, for the constant a1, if it is a penguin, it is also a bird, and if it is a bird, it also flies. It is conceivable that by iterating over this procedure, it is possible to also induce that for all the constants in the domain, if they are penguins, they do not fly. This would then give us the appropriate condition for constructing an abnormality predicate.

The applicability of the Markov logic network structure learner is not as immediate as ProbFOIL, but it is also not too distant from it. We suspect other types of rule learners, especially those relying on neural techniques (Bueff and Belle, 2024), could also be applied in a similar way to learn default theories.

It is perhaps less surprising to note that the idea of constructing logical exceptions is not new in itself. There have been a number of proposals that suggest an analogous framework to ours. For instance, Sakama (Sakama, 2005) suggests the use of negation as failure to construct rules of the following sort:

```
fly(x) :-
    bird(x),
    not pengiun(x), not crippled(x).
```

It might then be possible to construct an abnormality predicate from these exceptions.

Likewise, in early work, (Dimopoulos and Kakas, 1995) discusses the work of learning exceptions based on patterns in the negative examples to learn a hierarchical logic program. Analogously, in work that we became aware of at the time of writing this paper, the proposal of (Shakerin et al., 2017) is very similar in spirit. They propose an algorithm called FOLD – which has recently been extended to hybrid domains (Wang and Gupta, 2022) – that also aims to construct abnormality predicates based on exceptions. Thus, it sits in between the work of (Dimopoulos and Kakas, 1995) and ours. However, the semantics of this approach is tightly linked to and justified by their nonmonotonic semantical setup. In contrast, what we are able to demonstrate is that by using existing rule learners such as FOIL, ProbFOIL, or Markov Logic Network structure learners, it is simple to construct hierarchies of exceptions. This idea can be teased apart without committing to a specific language or even a specific type of exception learner. Perhaps because we use ProbFOIL, we might further add probabilities to defaults, which might enable the treatment of statistical defaults (Bacchus et al., 1996) in a practical manner.

In sum, from a semantical perspective, what we are attempting here is different from the previous work on learning defaults by appealing to one or more distinguished abnormality predicates. From an algorithmic point of view, we are able to leverage any structured learner because we essentially need to lump the exceptions as abnormal ones, yielding a default theory. Essentially, what we are proposing is a simple way of considering categories of objects under the abnormality predicate. We believe our approach is likely more accessible than previous accounts, such as (Schuurmans and Greiner, 1994).

6 CONCLUSION

We propose a simple approach to learning defaults by constructing abnormality predicates from all the negative instances in the learned rules. Such a model allows us to construct nested defaults as well. In contrast to most existing work, which either falls into the camp of being a conceptual model with limited evaluations or into the camp of specialized algorithms, the nature of our proposal is that it is accessible and can be built in principle on any structure.

The results of our quantitative analysis shows that learning speed is quicker on the dataset which contains the ab_pred data points alongside the normal data points, even though there is a higher number of data points in the learning data in the second stage than in the first stage. These findings seem to confirm that our system bears some resemblance to methods used for statistical predicate invention, and therefore has similar advantages. Specifically, it seems as though the effective addition of a new category enables the learning algorithm to find rules which explain the data more quickly.

For the future, it would be interesting to scale our results. Recall that the original ProbFOIL learning algorithm was evaluated (De Raedt et al., 2015) using the dataset from the NELL (Never-Ending Language Learning) project (Carlson et al., 2010). It would be interesting to run our algorithm on this dataset, compare it against the reported ProbFOIL results (De Raedt et al., 2015), and finally qualitatively evaluate the types of defaults induced.

An argument often made in the knowledge representation community (McCarthy and Hayes, 1969; McCarthy, 1980) is that allowing exceptions will likely make common sense reasoning easier without

stipulating all the exceptions explicitly. It is also widely acknowledged that defaults appear in many models about the world – for example, we likely frequently invoke *causal completeness* (Reiter, 1991) to reason about the physical world. Roughly, this means that a reasonable number of conditions capture the preconditions and the effects of actions, and those not mentioned are not relevant for the task at hand. It would be interesting to see whether our proposal here could allow the learning of such default theories, both in a static as well as a dynamic setting.

REFERENCES

Bacchus, F., Grove, A. J., Halpern, J. Y., and Koller, D. (1996). From statistical knowledge bases to degrees of belief. *Artificial intelligence*, 87(1-2):75–143.

Boutilier, C. (1994). Unifying default reasoning and belief revision in a modal framework. *Artificial Intelligence*, 68(1):33–85.

Bueff, A. and Belle, V. (2024). Learning explanatory logical rules in non-linear domains: a neuro-symbolic approach. *Machine Learning*, pages 1–36.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr, E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3.

De Raedt, L., Dries, A., Thon, I., Van den Broeck, G., and Verbeke, M. (2015). Inducing probabilistic relational rules from probabilistic examples. In *Proceedings of 24th international joint conference on artificial intelligence (IJCAI)*, pages 1835–1842.

De Raedt, L., Kimmig, A., and Toivonen, H. (2007). Problog: A probabilistic prolog and its application in link discovery.

Denecker, M., Marek, V. W., and Truszczynski, M. (2000). Uniform semantic treatment of default and autoepistemic logic. In *Proc. KR*, pages 74–84.

Dimopoulos, Y. and Kakas, A. (1995). Learning nonmonotonic logic programs: Learning exceptions. In *Machine Learning: ECML-95: 8th European Conference on Machine Learning Heraclion, Crete, Greece, April 25–27, 1995 Proceedings 8*, pages 122–137. Springer.

Etherington, D. W. (1987). Relating default logic and circumscription. In *Proceedings of the 10th international joint conference on Artificial intelligence - Volume 1*, pages 489–494, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Grosof, B. N. (1992). Representing and reasoning with defaults for learning agents. In *Proceedings of the ML92 Workshop on Biases in Inductive Learning. Proceedings Available from the Workshop Chair, Diana Gordon: Naval Research Laboratory, Washington, DC*, volume 20375.

Halpern, J. (1997). A Critical Reexamination of Default Logic, Autoepistemic Logic, and Only Knowing. *Computational Intelligence*, 13(1):144–163.

Kok, S. and Domingos, P. (2007). Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, pages 433–440. ACM.

Kok, S. and Domingos, P. (2010). Learning markov logic networks using structural motifs. In *ICML*, pages 551–558.

Lakemeyer, G. and Levesque, H. J. (2006). Towards an axiom system for default logic. In *Proc. AAAI*, pages 263–268.

McCarthy, J. (1980). Circumscription–a form of nonmonotonic reasoning. *Artificial intelligence*, 13(1-2):27–39.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502.

Morgenstern, L. and McIlraith, S. A. (2011). John McCarthy's legacy. *Artificial Intelligence*, 175(1):1 – 24.

Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., and Srinivasan, A. (2012). Ilp turns 20. *Machine learning*, 86(1):3–23.

Quinlan, J. R. and Cameron-Jones, R. M. (1993). Foil: A midterm report. In *Machine Learning: ECML-93: European Conference on Machine Learning Vienna, Austria, April 5–7, 1993 Proceedings 6*, pages 1–20. Springer.

Reiter, R. (1982). Circumscription implies predicate completion (sometimes). In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 418–420.

Reiter, R. (1991). The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial intelligence and mathematical theory of computation: Papers in honor of John McCarthy*, pages 359–380. Academic Press.

Sakama, C. (2005). Ordering default theories and nonmonotonic logic programs. *Theoretical Computer Science*, 338(1-3):127–152.

Schuurmans, D. and Greiner, R. (1994). Learning default concepts. In *Proceedings Of The Biennial Conference-Canadian Society For Computational Studies Of Intelligence*, pages 99–106.

Shakerin, F., Salazar, E., and Gupta, G. (2017). A new algorithm to automate inductive learning of default theories. *Theory and Practice of Logic Programming*, 17(5-6):1010–1026.

Valiant, L. (2013). Probably approximately correct: Nature's algorithms for learning and prospering in a complex world.

Valiant, L. G. (1999). Robust logics. In *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*, pages 642–651.

Wang, H. and Gupta, G. (2022). Fold-r++: a scalable toolset for automated inductive learning of default theories from mixed data. In *International Symposium on Functional and Logic Programming*, pages 224–242. Springer.