Logic-based Explainable and Incremental Machine Learning

Gopal Gupta
Huaduo Wang, Kinjal Basu, Farhad Shakerin, Elmer Salazar
Sarat Chandra Varanasi, Parth Padalkar, Sopam Dasgupta

Department of Computer Science
The University of Texas at Dallas, Richardson, USA

Abstract. Mainstream machine learning methods lack *interpretability*, *explainability*, *incrementality*, and *data-economy*. We propose using logic programming to rectify these problems. We discuss the FOLD family of rule-based machine learning algorithms that learn models from relational datasets as a set of default rules. These models are competitive with state-of-the-art machine learning systems in terms of accuracy and execution efficiency. We also motivate how logic programming can be useful for *theory revision* and *explanation based learning*.

1 Introduction

Dramatic success of machine learning has led to a plethora of artificial intelligence (AI) applications. The effectiveness of these machine learning systems, however, is limited in several ways:

1. **Lack of Interpretability:** The models learned by machine learning systems are opaque, i.e., they are not comprehensible by humans. This is mainly because these statistical machine learning methods produce models that are complex algebraic solutions to optimization problems such as risk minimization or likelihood maximization.
2. **Lack of Explainability:** These models are unable to produce a justification for a prediction they compute for a new data sample.
3. **Lack of Incrementality:** These methods are unable to incrementally update a learned model as new data is encountered.
4. **Lack of Data Economy:** These methods need large amounts of data to compute a model. Humans, in contrast, are able to learn from a small number of examples.

In this position paper we show that these problems are greatly alleviated if we develop machine learning methods that learn default theories coded in logic programming. The whole field of inductive logic programming (ILP) has been developed in which Horn clauses are learned from background knowledge, positive, and negative examples [8]. Rules with negated goals in the body are also learned in ILP as nonmonotonic logic programs and default rules [27, 11]. Representing a model as default rules brings significant advantages wrt interpretability, explainability, incremental learning, and data economy. We present LP-based machine learning algorithms that are interpretable and explainable, as well as LP-based reinforcement learning for incremental learning, and LP-based explanation based learning for solving data economy issues.

Default rules are an excellent way of capturing the logic underlying a relational dataset. Defaults are used by humans in their day-to-day reasoning [28, 10]. Most datasets are generated from human-driven activity (e.g., loan approval by bank officials) and our experiments indicate that the rules underlying the model learned from these datasets can be represented quite faithfully and succinctly with default rules. Default rules are used by humans to learn a concept in an elaboration tolerant manner, as they allow humans to constantly adjust the decision boundary. We have developed machine learning algorithms that learn default rules (the *model*) from relational data containing categorical (i.e., discrete) and numerical values that are competitive with state-of-the-art machine learning techniques. These algorithms are *interpretable* and *explainable*.

Once a set of default rules has been learned from data, it is possible that these rules may be wrong (possibly because we over-generalized or under-generalized). When human beings learn from examples (by formulating a rule of thumb in their mind), then when they encounter an example that goes against the learned rule, they revise the rule in light of this new example. For example, suppose we learn the rule that if object `X` is a fruit, then it goes into the refrigerator. Later, we learn from experience or someone may tell us that pineapples must not go into the refrigerator. In that case, we will revise the rule, changing it to: if `X` is a fruit, it goes into the refrigerator, except for pineapples. This is a form of *incremental* or reinforcement learning [1]. We will refer it to as logic-based reinforcement learning. Logic-based reinforcement learning can be regarded as theory revision. Logic-based reinforcement learning is elegantly modeled in logic programming using default theories as well.

Traditional machine learning methods need large amounts of data to learn. In contrast, humans can learn from a small number of examples. The problem of learning from a small number of examples has been explored under the topic of explanation-based learning (EBL) [16]. Explanation-based learning can be further developed and applied to practical applications within the framework of logic programming through the use of default theories.

Finally, knowledge expressed as a logic program can be incorporated in the neural learning process. Thus, logic programming can play an important role in neuro-symbolic learning [34, 17, 18]. However, we won't discuss this topic due to lack of space. Logic programming can make a significant difference in this area.

Note that we only give a brief overview of logic-based reinforcement learning and explanation-based learning. These techniques are really important for the field of machine learning, and logic programming can provide excellent solutions. Our hope is that the logic programming community will invest more effort in further developing them.

2 Default Rules

Default Logic [20] is a non-monotonic logic to formalize commonsense reasoning. A default D is an expression of the form

$$\frac{A : \mathbf{M}B}{\Gamma}$$

which states that the conclusion Γ can be inferred if pre-requisite A holds and B is justified. $\mathbf{M}B$ stands for "it is consistent to believe B". If we restrict ourselves to logic programming, then normal logic programs can encode a default theory quite elegantly [12]. A default of the form:

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n : \mathbf{M}\neg\beta_1, \mathbf{M}\neg\beta_2 \ldots \mathbf{M}\neg\beta_m}{\gamma}$$

can be formalized as the normal logic programming rule:

$$\gamma \texttt{ :- } \alpha_1, \alpha_2, \ldots, \alpha_n, \texttt{not } \beta_1, \texttt{not } \beta_2, \ldots, \texttt{not } \beta_m.$$

where α's and β's are positive predicates and `not` represents negation-as-failure. We call such rules *default rules*. Thus, the default

$$\frac{bird(X) : M\neg penguin(X)}{fly(X)}$$

will be represented as the following default rule in normal logic programming:

```
fly(X) :- bird(X), not penguin(X).
```

We call `bird(X)`, the condition that allows us to jump to the default conclusion that `X` can fly, the *default part* of the rule, and `not penguin(X)` the *exception part* of the rule.

3 Default Rules as Machine Learning Models

Default rules allow knowledge to be modeled in an *elaboration tolerant* manner [4, 12]. Default rules are an excellent vehicle for representing inductive generalizations. Humans indeed represent inductive generalizations as default rules [28, 10]. Arguably, the sophistication of the human thought process is in large part due to the copious use of default rules [12].

Consider the example about birds above. We observe that bird 1 can fly, bird 2 can fly, bird 3 can fly, and so on. From this we can generalize and learn the default rule that "birds fly." But then we notice that a few of the birds that are penguins, do not fly. So we add an exception to our rule: "birds fly, unless they are penguins". What we are doing is really adjusting our decision boundary, as illustrated in Fig. 1(i) and Fig. 1(ii) (black dots represents normal birds, red dots represent penguins). In logic programming, we can make the exception part of the rule explicit, and code it as:

```
fly(X) :- bird(X), not abnormal_bird(X).
abnormal_bird(X) :- penguin(X).
```

Suppose, we later discover that there is a subclass of penguins (called super-penguins [11]) that can fly. In such a case, we have learned an exception to an

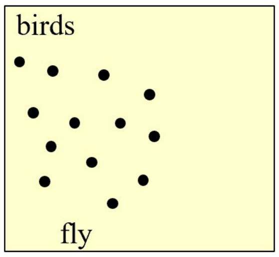

Fig. 1(i)

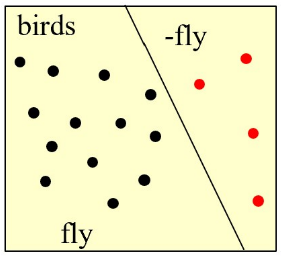

Fig. 1(ii)

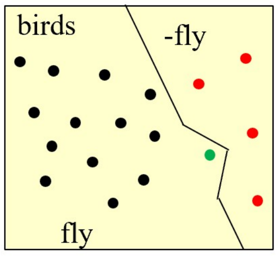

Fig. 1(iii)

exception (See Fig. 1(iii); green dot represents a super-penguin). This will be coded in logic programming as:

```
fly(X) :- bird(X), not abnormal_bird(X).
abnormal_bird(X) :- penguin(X), not abnormal_penguin(X).
abnormal_penguin(X) :- superpenguin(X).
```

Thus, default rules with exceptions, exceptions to exceptions, exceptions to exceptions to exceptions, and so on, allow us to dynamically refine our decision boundary as our knowledge of a concept evolves. This is the insight that the FOLD family of algorithms uses to learn a model underlying a dataset. Figure 1 illustrates the idea.

3.1 FOLD Family of Machine Learning Algorithms

We have developed the FOLD family of algorithms that output a model represented as default rules. Our inspiration is the FOIL algorithm of Quinlan [19]. The FOLD algorithm is a top-down rule-learning algorithm [26]. It starts with the candidate rule

```
p(X,L) :- true.
```

where `p(X,L)` is the target predicate to learn, and states that record `X` has the label `L` (for example, for the Titanic survival dataset, the target predicate will be `status(Passenger_Id, S)`, where `S` is either `survived` or `perished`). It then extends the body with a selected literal (predicate) from among the features so as to cover maximum number of positive examples and avoid covering maximum number of negative examples. The process of selecting a literal to add to the body of the rule relies on heuristics. Traditionally, the Information Gain (IG) heuristic has been used [19, 26]. The IG heuristic was pioneered in the FOIL algorithm [19] for learning logic programs and adapted by us for the FOLD algorithm [26] to learn default theories. The FOLD algorithm learns a default theory, so in the next step of learning, it swaps the remaining uncovered positive and negative examples, and recursively applies the literal selection step to learn the exception to the default. Literal selection with swapping of uncovered positive and negative examples continues until reasonable accuracy is obtained. Thus, given the following information, represented as predicates:

```
bird(tweety).          bird(woody).
cat(kitty).            penguin(polly).
bird(polly).
```

and given positive examples:

```
     E+: fly(tweety).          fly(woody).
```
and negative examples
```
     E-: fly(polly).           fly(kitty).
```
FOLD will learn the default theory:
```
     fly(X) :- bird(X), not abnormal(X).
     abnormal(X) :- penguin(X).
```
The FOLD algorithm inspired a family of algorithms for learning stratified normal logic programs: LIME-FOLD [24], SHAP-FOLD [25], FOLD-R [23], FOLD-R++ [29], FOLD-RM [32], FOLD-SE [30] and FOLD-TR [31]. Various algorithms in this family differ in the heuristic used for literal selection as well as how efficiently the heuristic is computed. Given a labeled training dataset, these learning algorithms learn a default theory that serves as a model. The default theory is represented as a stratified normal logic program, and hence is *interpretable*. Given a new data record, the model will predict the outcome by executing the logic program. The proof tree generated during execution serves as an *explanation* for the decision reached by the model.

For the FOLD algorithm family, the dataset can have numerical and categorical features, however, the classification label should be categorical. LIME-FOLD is based on using the LIME method [21] for determining the level of contribution of each feature to a prediction [24]. SHAP-FOLD uses Shapley values [15] instead. FOLD-R and FOLD-R++ are binary classifiers that use information gain [19] as the heuristic, where FOLD-R++ uses prefix sums to speed up the computation. FOLD-RM is a multi-class classifier version of FOLD-R++. FOLD-SE is based on a new heuristic called Gini Impurity [14]. This new heuristic leads to a significant reduction in the number of learned default rules and literals. FOLD-SE provides *scalable interpretabilty*, in that the number of default rules learned does not increase with the size of the dataset. The number of rules learned may be as small as 2 for a highly accurate model given datasets of sizes as large as 150,000. FOLD-TR [31] uses the FOLD algorithm to learn to rank.

3.2 Examples and Performance

We next give an example. The Titanic survival prediction is a classical classification challenge, which contains 891 passengers as training examples and 418 passengers as testing examples. The default rule-set learned by our FOLD algorithm is shown below. It captures the underlying logic of the model, namely, that female infant passengers, aged less than 5 in 3rd class who paid fare less than or equal to 10.463 units, perished. Male passengers perished unless they paid a fare in the range 26.25 to 26.387 units inclusive *or* they were younger than 12 with 2 or fewer relatives and were not without parents.

```
status(X,perished) :- class(X,'3'), not sex(X,'male'),
           age(X,N1), N1=<5.0, fare(X,N4), N4=<10.463.
status(X,perished) :- sex(X,'male'), not ab1(X), not ab3(X).
ab1(X) :- fare(X,N4), N4>26.25, N4=<26.387.
ab2(X) :- number_of_parents_children(X,N3), N3=<0.0,
          age(X,N1),N1=<11.0.
```

```
ab3(X) :- age(X,N1),N1=<12.0, number_of_siblings_spouses(X,N2),
          N2=<2.0,  not ab2(X).
status(X,survived) :- not status(X,perished).
```

The rules represent a default theory with (nested) exceptions (`ab1, ab2, ab3`). If a person did not perish, they survived. The model's accuracy is 0.98, precision is 1.0, recall is 0.96, and F1-score is 0.98 [1]. These rules are interpretable by a human.

The learned program can be executed in a Prolog system or an answer set programming system such as s(CASP) [3]. The s(CASP) system can also generate a proof tree that serves as an explanation for a prediction made for a given input. The FOLD-SE system itself also generates an explanation. Explainability is important for understanding the prediction made by a machine learned model. The FOLD family of algorithms are comparable in accuracy to state-of-the-art machine learning systems such as XGBoost [6] and Multi-Layer Perceptrons (MLPs) [1]. XGBoost is a very popular machine learning method based on gradient boosting, while MLPs are neural networks. The FOLD family of algorithms is an order of magnitude faster than XGBoost and MLPs, and, in addition, is explainable. The FOLD family algorithms perform minimal pre-processing of the dataset (no need for one-hot encoding [1], for example). Table 1 shows the performance of FOLD-SE compared to the most prominent rule-based machine learning algorithm called RIPPER [7] on selected datasets (a more extensive comparison can be found elsewhere [30]).

We can see that FOLD-SE outperforms RIPPER on the number of rules generated and execution time. The number of rules may go down from approximately 180 to between 2 and 3 (for rain in Australia dataset, for example). The results reported are an average over 10 runs, so the number of rules reported in the table can be fractional. The scalability of FOLD-SE with respect to interpretability, namely, the number of rules generated is small regardless of the size of the dataset, shows the power of representing inductive generalizations as default rules.

Data Set			RIPPER					FOLD-SE				
Name	Rows	Cols	Acc	F1	T(ms)	Rules	Preds	Acc	F1	T(ms)	Rules	Preds
acute	120	7	0.93	0.92	95	**2.0**	4.0	**1.0**	**1.0**	**1**	**2.0**	**3.0**
heart	270	14	0.76	0.77	317	5.4	12.9	0.74	0.77	**13**	**4.0**	**9.1**
breast-w	699	10	0.93	0.90	319	14.4	19.9	0.94	0.92	**9**	**3.5**	**6.3**
eeg	14980	15	0.55	0.36	12,996	43.4	134.7	0.67	0.68	**1,227**	**5.1**	**12.1**
cr. card	30000	24	0.76	0.84	49,940	36.5	150.7	**0.82**	**0.89**	**3,513**	**2.0**	**3.0**
adult	32561	15	0.71	0.77	63,480	41.4	168.4	**0.84**	**0.90**	**1,746**	**2.0**	**5.0**
rain in aus	145460	24	0.63	0.70	3118,025	180.1	776.4	**0.82**	**0.89**	**10,243**	**2.5**	**6.1**

Table 1. Comparison of RIPPER and FOLD-SE on selected Datasets

Table 2 compares the performance of FOLD-SE with state-of-the-art machine learning tools XGBoost and Multilayer Perceptrons (MLPs) on training time (in milliseconds). Note that Accuracy and F1-score that are standard metrics [1] are reported. Table dimension is also given (Rows x Columns). The best

Data Set			XGBoost			MLP			FOLD-SE		
Name	Rows	Cols	Acc	F1	T(ms)	Acc	F1	T(ms)	Acc	F1	T(ms)
acute	120	7	**1.0**	**1.0**	122	0.99	0.99	22	**1.0**	**1.0**	**1**
heart	270	14	**0.82**	**0.83**	247	0.76	0.78	95	0.74	0.77	**13**
breast-w	699	10	0.95	0.96	186	**0.97**	**0.98**	48	0.94	0.92	**9**
eeg	14980	15	0.64	**0.71**	46,472	**0.69**	**0.71**	9,001	0.67	0.68	**1,227**
credit card	30000	24	NA	NA	NA	NA	NA	NA	**0.82**	**0.89**	**3,513**
adult	32561	15	**0.87**	**0.92**	424,686	0.81	0.87	300,380	0.84	0.90	**1,746**
rain in aus	145460	24	**0.84**	**0.90**	385,456	0.81	0.88	243,990	0.82	0.89	**10,243**

Table 2. Comparison of XGBoost, MLP, and FOLD-SE

performer is highlighted in bold. FOLD-SE is comparable in accuracy to widely used machine learning systems such as XGBoost and MLPs, yet it is an order of magnitude faster and is explainable. Thus, the default rule representation of machine learning models is quite effective. Note that the FOLD algorithms have been extensively compared with other ILP methods that learn answer set programs elsewhere [32, 29, 30, 25]. We do not repeat the comparison here due to lack of space. FOLD-SE outperforms Decision Trees [1] in terms of brevity of explanation. For example, for the adult dataset, the tree generated in the Decision Tree method has 4000+ nodes, around half of which are leaf nodes. This translates into 2,000 odd decision rules. Also, the average depth of leaf nodes is 24.7. Therefore, there would be around 50,000 predicates in the decision rule-set. The FOLD-SE algorithm, in contrast, only generates 2 rules with 5 predicates, while achieving greater accuracy.

Default rules can also be used to make convolutional neural networks (CNNs) explainable as done in the NeSyFOLD system [18], for example (CNNs are neural networks designed for learning from image data).

4 Logic-based Incremental Learning

The FOLD family of algorithms permits us to learn explainable theories from positive and negative examples. Once a theory is learned, we may encounter further instances (examples) and we may attempt to explain them using this learned theory. If we succeed, our beliefs get stronger, however, if we fail, we update our beliefs to accommodate the new example. For instance, let's say a child drops a glass object on the floor. The object shatters. After a few such mishaps, the child quickly learns the rule "a glass object dropped on the floor will shatter" and will be careful afterwards when holding glass objects. However, later, the child drops another glass object, but the object falls on a soft surface (e.g., carpeted surface), and does not break. The child then updates the prior belief to "a glass object dropped on the floor will shatter, unless the floor is carpeted".

This action/reward-based learning technique of humans closely relates to Reinforcement Learning (RL) [1] in the realm of Machine Learning. In RL, while exploring an environment, an agent gets positive/negative rewards based on its actions. From these rewards, the agent may learn—and subsequently revise—a *policy* to take better actions in its operating environment. The policy can be represented symbolically as a default rule-set, and continuously refined. To achieve this we need a *theory revision framework* that can be applied to default rules. We have developed such an incremental learning framework [5] that we summarize next. This work is distinct from our work on FOLD family of algorithms described earlier. Note that incremental learning is performed by revising the existing theory through the use of the s(CASP) goal-directed answer set programming (ASP) system [3]. With the s(CASP) system, we can obtain a proof tree for any reasoning task performed. This tree can be analyzed and used to update the rules. We illustrate our incremental learning framework through an example.

Consider a house that has sensors installed to protect it from fires and floods. To protect from fire, a fire sensor is installed that will automatically turn on water sprinklers installed in the house if fire is detected. Likewise, a water leak detection sensor in the house will automatically turn off water supply, if water is detected on the floor/carpet and no one is present in the house. The following logic program models these rules:

```
fireDetected :- fire.
turnSprinklerOn :- fireDetected.
sprinklerOn :- turnSprinklerOn.
water :- sprinklerOn.

sprinklerOff :- waterSupplyOff.
waterSupplyOff :- turnWaterSupplyOff.
turnWaterSupplyOff :- houseEmpty, waterLeakDetected.
waterLeakDetected :- water.

houseFloods :- water, not waterSupplyOff.
houseBurns :- fireDetected, SprinklerOff.
houseSafe :- not houseFloods, not houseBurns.
```

The program is self-explanatory, and models fluents (`sprinklerOn`, `sprinklerOff`, `waterLeakDetected`, `fireDetected`, `fire`, `water`, `houseEmpty`) and actuators (`turnWaterSupplyOff`, `turnSprinklerOn`). The fluents `fire`, `water`, and `houseEmpty` correspond to sensors. For simplicity, time is not considered. Note that 'fire' means fire broke out in the house and 'water' means that a water leak occurred in the house.

Given the theory above, if we add the fact `fire.` to it, we will find that the property `houseBurns` defined above will succeed. This is because the occurrence of fire eventually leads to sprinklers being turned on, which causes water to spill on the floor, which, in turn, causes the flood protection system to turn on, and turn off the water supply. We want `houseBurns` to fail. To ensure that it fails, we have to recognize that the water supply should be turned off due to a water leak in an empty house *unless* the house is on fire:

```
turnWaterSupplyOff :- houseEmpty, waterLeakDetected,
                                          not fireDetected.
```

Therefore, a simple patch to the theory shown above will ensure that houseBurns fails in all situations. By adding *not fireDetected* we are subtracting knowledge preventing `houseBurns` from succeeding.

Note that to solve the theory revision problem we should be able to analyze the resulting search tree both when a query succeeds or when it fails. The s(CASP) system provides access to the search tree through the justification tree it produces [2]. For a failed query, we obtain the (successful) search tree of the negated query, and analyze its corresponding justification tree. There has been a significant amount of work done on theory revision (or knowledge refinement) in the context of logic [22]. Most of this work assumes that the theory is expressed using Horn clauses. Adding negation-as-failure through answer set programming (along with the ability to obtain proof-trees for queries through s(CASP)) allows for a more powerful theory revision framework [5]. This work is a first step towards building more powerful theory revision frameworks based on using logic programming, specifically, goal-directed answer set programming.

5 Explanation-based Learning

Machine learning algorithms require large amounts of data to learn a model. Ideally, a small amount of data should be sufficient for learning, including learning or generalizing from just *one example*. In order to do so, however, background (commonsense) knowledge is essential. This aspect is important in machine learning and has been explored under the topic of *explanation-based learning (EBL)*. Logic programming can play an important role in EBL. EBL has been extensively investigated in the past [9, 33].

EBL essentially uses prior knowledge to"explain" each training example. An explanation identifies properties that are relevant to a target concept. EBL trades a large number of examples needed in traditional machine learning with background knowledge (called domain theory) for the target concept. The domain theory should be correct (no negative examples entailed), complete (all positive examples are covered), and tractable (each positive example can be explained). It must be noted that EBL has been viewed as a variation of partial evaluation [13]. Traditionally, EBL takes as input a set of training examples, a domain theory expressed using Horn clauses, and operationality criteria that restrict the hypothesis space to a fixed set of predicates, e.g., those needed to directly describe the examples. The goal is to find an efficient definition of the target concept, consistent with both the domain theory and the training examples. Let's consider a very simple example about stacking one object on another. Suppose we have the following specific example involving two objects `obj1` and `obj2`.

```
safeToStack(obj1,obj2).    on(obj1,obj2).    owner(obj1,molly).
type(obj2,endtable).       type(obj1,box).   owner(obj2, john).
fragile(obj2).             color(obj1,red).  color(obj2, blue).
material(obj1,cardboard).  material(obj2, wood).
volume(obj1, 2).         density(obj1, 0.1).
```

and the domain theory explaining why it is safe to stack `obj1` on `obj2`

```
safeToStack(obj1,obj2) :- lighter(obj1,obj2).
lighter(obj1,obj2) :- weight(obj1, W1), weight(obj2,W2), W1 < W2.
weight(X, W) :- volume(X, V), density(X,D),  W =:= V*D.
weight(X,5) :- type(X, endtable).
```

The operational predicates are `type, volume, density, on, <, >, =:=`. Partially evaluating the predicate call `safeToStack(obj1,obj2)` against the domain theory while keeping only the operational predicates and generalizing will allow us to learn the general rule:

```
   safeToStack(X,Y) :- volume(X,V), density(X,D), WX =:= V*D,
                       type(Y, endtable), WX < 5.
```

Essentially, we have learned a rule that tells us when two objects can be safely stacked, defined in terms of properties of the two objects. While this may appear as simple partial evaluation, more intelligent reasoning can be incorporated by bringing in additional background knowledge. For example, we know that an end-table is similar to a center-table with respect to the stacking property, and so we may further generalize our rule to work for more types of tables. We could generalize it even further to any heavy table-like structure with a flat top, and so on.

EBL can be made more powerful and flexible by representing the domain theory in ASP. For example, constraints can be stated to block simplification along certain evaluation paths during partial evaluation. More general generalizations—represented as default rules—can be made that also account for exceptions. In the example above, while generalizing, we may want to rule out tables that have small-sized surface top, as there is a danger of tipping over upon stacking. We may also want to avoid stacking on tables made of fragile material (another exception). Of course, more knowledge about a table's dimensions, the material it is made of, etc., will have to be added to achieve this. Generalizing to ASP, however, would require developing a partial evaluator for answer set programming under some operational semantics. The s(CASP) system provides one such operational semantics.

6 Conclusion

Default rules are an elegant way to represent knowledge underlying machine learning models. We have developed very efficient machine learning algorithms (e.g., FOLD-SE described above) that are competitive with state-of-the-art methods. Likewise, knowledge represented as default rules can be incrementally updated—via theory revision—as well as generalized/specialized—via explanation-based learning. It is our position that logic-based approaches—centered on representing knowledge as default theories—can lead to the design of machine learning systems that can be competitive with state-of-the-art traditional machine learning systems, while providing advantages of interpretability, explainability, incrementality, and data economy.

Acknowledgements: We are grateful to anonymous reviewers and to Bob Kowalski for insightful comments that helped in significantly improving this paper. Authors acknowledge partial support from NSF grants IIS 1910131, IIP 1916206, and US DoD.

References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning - A Textbook.* Springer, 2018.

[2] Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. "Justifications for Goal-Directed Constraint Answer Set Programming". In: *Proceedings 36th International Conference on Logic Programming (Technical Communications).* Vol. 325. EPTCS. 2020, pp. 59–72.

[3] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. "Constraint Answer Set Programming without Grounding". In: *TPLP* 18.3-4 (2018), pp. 337–354.

[4] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press, 2003.

[5] Kinjal Basu et al. "Symbolic Reinforcement Learning Framework with Incremental Learning of Rule-based Policy". In: *Proc. ICLP GDE'22 Workshop.* Vol. 3193. CEUR Workshop Proceedings. CEUR-WS.org, 2022.

[6] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD.* KDD '16. San Francisco, California, USA, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2.

[7] William W. Cohen. "Fast Effective Rule Induction". In: *Proc. ICML.* San Francisco, CA, USA, 1995, 115–123.

[8] Andrew Cropper and Sebastijan Dumancic. *Inductive logic programming at 30: a new introduction.* arXiv:2008.07912. 2020.

[9] Gerald DeJong and Raymond J. Mooney. "Explanation-Based Learning: An Alternative View". In: *Mach. Learn.* 1.2 (1986), pp. 145–176.

[10] E.A. Dietz Saldanha, S. Hölldobler, and L.M. Pereira. "Our Themes on Abduction in Human Reasoning: A Synopsis". In: *Abduction in Cognition and Action: Logical Reasoning, Scientific Inquiry, and Social Practice.* 2021, pp. 279–293.

[11] Yannis Dimopoulos and Antonis Kakas. "Learning non-monotonic logic programs: Learning exceptions". In: *Machine Learning: ECML-95.* Ed. by Nada Lavrac and Stefan Wrobel. Berlin, Heidelberg, 1995, pp. 122–137.

[12] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press, 2014.

[13] Frank van Harmelen and Alan Bundy. "Explanation-Based Generalisation = Partial Evaluation". In: *Artif. Intell.* 36.3 (1988), pp. 401–412.

[14] Eduardo Laber, Marco Molinaro, and Felipe Mello Pereira. "Binary Partitions with Approximate Minimum Impurity". In: *Proc. ICML.* Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2854–2862.

[15] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in Neural Information Processing Systems.* 2017, pp. 4765–4774.

[16] Steven Minton et al. “Explanation-Based Learning: A Problem Solving Perspective”. In: *Artif. Intell.* 40.1-3 (1989), pp. 63–118.

[17] Ludovico Mitchener, David Tuckey, Matthew Crosby, and Alessandra Russo. “Detect, Understand, Act: A Neuro-symbolic Hierarchical Reinforcement Learning Framework”. In: *Mach. Learn.* 111.4 (2022), pp. 1523–1549.

[18] Parth Padalkar, Huaduo Wang, and Gopal Gupta. *NeSyFOLD: A System for Generating Logic-based Explanations from Convolutional Neural Networks.* arXiv:2301.12667. 2023.

[19] J. Ross Quinlan. “Learning Logical Definitions from Relations”. In: *Machine Learning* 5 (1990), pp. 239–266.

[20] Ray Reiter. “A logic for default reasoning”. In: *Artificial Intelligence* 13.1-2 (1980), pp. 81–132.

[21] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proc. KDD.* ACM, 2016, pp. 1135–1144.

[22] Bradley L. Richards and Raymond J. Mooney. “Automated Refinement of First-Order Horn-Clause Domain Theories”. In: *Machine Learning* 19.2 (1995), pp. 95–131.

[23] Farhad Shakerin. “Logic Programming-based Approaches in Explainable AI and Natural Language Processing”. Department of Computer Science, The University of Texas at Dallas. PhD thesis. 2020.

[24] Farhad Shakerin and Gopal Gupta. “Induction of Non-Monotonic Logic Programs to Explain Boosted Tree Models Using LIME”. In: *Proc. AAAI.* AAAI Press, 2019, pp. 3052–3059.

[25] Farhad Shakerin and Gopal Gupta. *Induction of Non-Monotonic Rules From Statistical Learning Models Using High-Utility Itemset Mining.* arXiv:1905.11226. 2019.

[26] Farhad Shakerin, Elmer Salazar, and Gopal Gupta. “A new algorithm to automate inductive learning of default theories”. In: *TPLP* 17.5-6 (2017), pp. 1010–1026.

[27] Ashwin Srinivasan, Stephen H. Muggleton, and Michael Bain. “Distinguishing Exceptions From Noise in Non-Monotonic Learning”. In: Proc. International Workshop on Inductive Logic Programming, 1992.

[28] Keith Stenning and Michiel van Lambalgen. *Human Reasoning and Cognitive Science.* MIT Press, 2008.

[29] Huaduo Wang and Gopal Gupta. “FOLD-R++: A Scalable Toolset for Automated Inductive Learning of Default Theories from Mixed Data”. In: *Functional and Logic Programming: 16th International Symposium, FLOPS 2022.* Kyoto, Japan: Springer-Verlag, 2022, 224–242. ISBN: 978-3-030-99460-0.

[30] Huaduo Wang and Gopal Gupta. *FOLD-SE: Scalable Explainable AI.* 2022.

[31] Huaduo Wang and Gopal Gupta. *FOLD-TR: A Scalable and Efficient Inductive Learning Algorithm for Learning To Rank*. 2022. arXiv: 2206.07295.

[32] Huaduo Wang, Farhad Shakerin, and Gopal Gupta. "FOLD-RM: Efficient Scalable Explainable AI". In: *TPLP* 22.5 (2022), pp. 658–677.

[33] Judith Wusteman. "Explanation-Based Learning: A survey". In: *Artif. Intell. Rev.* 6.3 (1992), pp. 243–262.

[34] Zhun Yang, Adam Ishay, and Joohyung Lee. "NeurASP: Embracing Neural Networks into Answer Set Programming". In: *IJCAI 2020*. Ed. by Christian Bessiere. 2020, pp. 1755–1762.