CSE 505 - Computing with Logic
Project - Phase 2

Charuta Pethe
111424850

November 15, 2017

1 Introduction

The paper that I have chosen is *A New Algorithm to Automate Inductive Learning of Default Theories* [1], by Farhad Shakerin, Elmer Salazar and Gopal Gupta from The University of Texas at Dallas, Texas, USA. This paper was published in the 33rd International Conference on Logic Programming (ICLP) at Melbourne, Australia in August 2017.

2 Paper Methodology

2.1 Background

Classical machine learning methods involve predictive models that are algebraic solutions to optimization problems. These methods are neither intuitive nor easily understandable, and are hence hard to justify when applied to a new data sample. Moreover, upon addition of new knowledge, the entire model needs to be relearned.

However, Inductive Logic Programming (ILP) [2] is a technique where the model is in the form of Horn clauses that are more comprehensible, and also allow the knowledge base to be incrementally extended without having to relearn the entire model. However, as negation-as-failure cannot be represented using Horn clauses, ILP is insufficient for reasoning in the absence of complete background knowledge. In addition, ILP cannot handle exceptions to rules, which results in exceptions being treated the same as noise.

The exceptions to rules often follow a pattern themselves, and can be learned. The resulting theory that is learned is a default theory, which describes the underlying model more accurately, and is more intuitive and comprehensible.

The paper presents two algorithms for learning default theories:

1. FOLD (First Order Learner of Default) - To handle categorical features
2. FOLD-R - To handle numeric features.

2.2 Problem Statement

The problem that is tackled in the paper can be formalized as follows:

Given

- a background theory ***B*** in the form of a normal logic program, i.e clauses of the form
  h ← *l1, ..., lm, not lm+1, ..., not ln*, where *h* and *l1, ..., ln* are positive literals
- two disjoint sets of grounded goal predicates ***E+*** and ***E-*** (positive and negative examples respectively)
- a hypothesis language of predicates ***L***
- a function ***covers****(H, E, B)* which returns the subset of E that is extensionally implied by the current hypothesis H, given the background knowledge B

Find

- a theory ***T***, for which *covers(T, E+, B)* = *E+* and *covers(T, E-, B)* = ϕ

2.3 FOLD Algorithm

The FOLD algorithm learns a concept as a default theory, along with exceptions. A brief description of the steps in the algorithm is as follows:

1. FOLD first tries to learn the default theory by specializing a general rule of the form $goal(V_1, ..., V_n) \leftarrow true$.
2. Each specialization rules out some already covered negative examples without decreasing the number of positive examples covered significantly.
3. This process stops once the information gain (as explained in Section 2.5) becomes zero.
4. If some negative examples are still covered at this point, they are either noisy data samples or exceptions to the rule learned so far. Finding the set of rules governing the set of negative examples is now a subproblem.
5. To solve this subproblem, FOLD swaps the current positive and negative examples, and recursively calls itself to learn the exception rules.
6. Each time a rule is discovered for a set of exceptions, a new predicate is introduced, and the negation of the predicate is added to the original rule.
7. If there is noisy data or uncertainty due to lack of information, there is no pattern to learn. In this case, FOLD enumerates the positive examples.

The paper proposes the following proofs about the algorithm:

1. **Finiteness:** The FOLD algorithm terminates on any finite set of examples.
2. **Soundness:** The FOLD algorithm always learns a hypothesis that covers no negative examples.
3. **Completeness:** The FOLD algorithm always learns a hypothesis that covers all positive examples.

2.3.1 FOLD: Example 1

Input

B: $bird(X) \leftarrow penguin(X).\ bird(tweety).\ bird(coco).\ cat(kitty).\ penguin(polly).$
Goal: To learn the rule $fly(X)$.
E+: $\{tweety,\ coco\}$
E-: $\{kitty,\ polly\}$

Output

$fly(X) \leftarrow bird(X),\ not\ ab0(X).$
$ab0 \leftarrow penguin(X).$

2.3.2 FOLD: Example 2

FOLD can also learn the correct theory in presence of nested exceptions.

Input

B: $bird(X) \leftarrow penguin(X).\ penguin(X) \leftarrow superpenguin(X).$
$bird(a).\ bird(b).\ penguin(c).\ penguin(d).\ superpenguin(e).\ superpenguin(f).$
$cat(c1).\ plane(g).\ plane(h).\ plane(k).\ plane(m).\ damaged(k).\ damaged(m).$
Goal: To learn the rule $fly(X)$.
E+: $\{a,\ b,\ e,\ f,\ g,\ h\}$
E-: $\{c,\ d,\ c1,\ k,\ m\}$

Output

$fly(X) \leftarrow plane(X),\ not\ ab0(X).$
$fly(X) \leftarrow penguin(X),\ not\ ab1(X).$
$fly(X) \leftarrow superpenguin(X).$
$ab0 \leftarrow damaged(X).$
$ab1 \leftarrow penguin(X).$

2.4 FOLD-R Algorithm

The FOLD-R algorithm is a numeric extension of the FOLD algorithm, which can be applied to data sets containing a mix of categorical and numerical data. Instead of performing discretization of the numeric data to qualitative values which leads to accuracy loss, this algorithm uses the same approach that is taken in the C4.5 algorithm.

For a numeric feature A, constraints such as {A ≤ h, A > h} have to be considered, where the threshold *h* is found by sorting the values of A and choosing the split that maximizes the information gain.

The steps in the FOLD-R algorithm are similar to the steps in the FOLD algorithm, except for a few changes made in order to incorporate numeric features. They are as follows:

1. In the specialization of the current clause, the best categorical variable, i.e. the one which maximizes information gain(as explained in Section 2.5), is chosen.
2. Similarly, the best numeric literal along with the best arithmetic constant and threshold with the highest information gain is chosen.
3. If neither the best categorical literal nor the best numeric literal leads to a positive information gain, then FOLD-R tries to find exceptions to the rule. If no exceptions can be found, the algorithm enumerates all the remaining examples.
4. If the information gain of the best categorical literal is more than that of the best numeric literal, then the clause of the categorical literal is chosen as the specialized clause of the current iteration.
5. If the information gain of the best numeric literal is more than that of the best categorical literal, then the clause of the numeric literal along with the arithmetic constraint is chosen as the specialized clause of the current iteration.

2.4.1 FOLD-R: Example

Input
The input data shown in Table 1 contains data with numeric as well as categorical features. "Temperature" and "Humidity" are numeric features, while "Outlook" and "Wind" are categorical features.

Table 1: FOLD-R Example: Input Data

Outlook	**Temperature**	**Humidity**	**Wind**	**PlayTennis**
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	83	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

Output
$play(X) \leftarrow overcast(X).$
$play(X) \leftarrow temperature(X, A),\ A \leq 75,\ not\ ab0(X).$
$ab0 \leftarrow windy(X),\ rainy(X).$
$ab0 \leftarrow humidity(X, A),\ A \geq 95,\ sunny(X).$

The algorithm suggests that an abnormal day to play is either a rainy and windy day or a sunny day with humidity above 95%.

2.5 Information Gain

As explained in Sections 2.3 and 2.4, the literal which has the maximum information gain is chosen and added to the set of rules. It is calculated as follows:

$$IG(L,R) = t\left(log_2\frac{p_1}{p_1+n_1} - log_2\frac{p_0}{p_0+n_0}\right)$$

Here:
L is the candidate literal to add to rule R
p_0 is the number of positive examples implied by the rule R
n_0 is the number of negative examples implied by the rule R
p_1 is the number of positive examples implied by the rule $R+L$
n_1 is the number of negative examples implied by the rule $R+L$
t is the number of positive examples implied by the rule R and covered by the rule $R+L$

3 Project Roadmap

I plan to implement, test and verify the algorithms presented in this paper. The steps in my plan to implement the project are as follows:

1. Implement FOLD in Python
2. Implement C4.5 in Python
3. Implement FOLD-R in Python
4. Create my own examples to test FOLD and FOLD-R
5. List the ideal outputs for my examples
6. Run the algorithms on my examples
7. Compare ideal outputs against experimental outputs
8. Note the results and analyze them (more specifically, analyze the cases in which the experimental outputs differ from the ideal outputs, and try to answer why).

References

[1] Farhad Shakerin, Elmer Salazar and Gopal Gupta, *A New Algorithm to Automate Inductive Learning of Default Theories*, International Conference on Logic Programming (ICLP), Melbourne, 2017.

[2] Stephen Muggleton, *Inductive Logic Programming*, New Generation Computing, 8, 295-318, 1991.