Whitebox Induction of Default Rules Using High-Utility Itemset Mining

Farhad Shakerin(✉) and Gopal Gupta

The University of Texas at Dallas, Richardson, TX 75080, USA
{farhad.shakerin,gopal.gupta}@utdallas.edu

Abstract. We present a fast and scalable algorithm to induce *non-monotonic* logic programs from statistical learning models. We reduce the problem of search for best clauses to instances of the *High-Utility Itemset Mining* (HUIM) problem. In the HUIM problem, feature values and their importance are treated as transactions and utilities respectively. We make use of TreeExplainer, a fast and scalable implementation of the Explainable AI tool SHAP, to extract locally important features and their weights from ensemble tree models. Our experiments with UCI standard benchmarks suggest a significant improvement in terms of classification evaluation metrics and training time compared to ALEPH, a state-of-the-art *Inductive Logic Programming* (ILP) system.

Keywords: Inductive logic programming · Machine learning · Explainable AI · Negation as failure · Answer set programming · Data mining

1 Introduction

The FOIL algorithm by Quinlan [14] is a popular ILP algorithm that incorporates heuristics from information theory called *weighted information gain* to guide the search for best clauses. The use of a greedy heuristic makes FOIL fast and scalable. However, scalability comes at the expense of losing accuracy if the algorithm is stuck in a local optima and/or when the number of examples is insufficient. Figure 1 demonstrates how the local optima results in discovering sub-optimal rules that does not necessarily coincide with the real underlying sub-concepts of the data.

Unlike FOIL, statistical machine learning algorithms are bound to find the relevant features because they optimize an objective function with respect to global constraints. This results in models that are inherently complex and cannot explain what features account for a classification decision on any given data sample. The Explainable AI techniques such as LIME [15] and SHAP [10] have been proposed that provide explanations for any given data sample. Each explanation is a set of feature-value pairs that would locally determine what features

Authors are partially supported by NSF Grants IIS 1718945 and IIS 1910131.

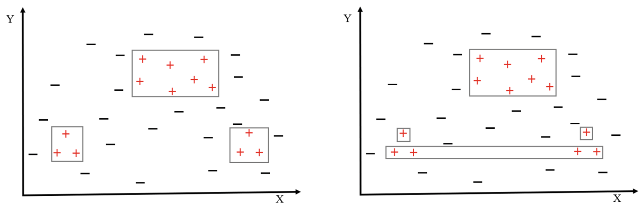

Fig. 1. Optimal sequential covering with 3 Clauses (Left), Sub-Optimal sequential covering with 4 Clauses (Right)

and how strongly each feature, relative to other features, contributes to the classification decision. To capture the global behavior of a black-box model, however, an algorithm needs to group similar data samples (i.e., data samples for which the same set of feature values are responsible for the choice of classification) and cover them with the same clause. While in FOIL, the search for a clause is guided by heuristics, in our novel approach, we adapt *High Utility Item-set Mining* (HUIM) [5]—a popular technique from data mining—to find clauses. We call this algorithm SHAP-FOLD from here on. The advantage of SHAP-FOLD over heuristics-based algorithms such as FOIL is that:

1. SHAP-FOLD does not get stuck in a local optima
2. SHAP-FOLD distinguishes exceptional cases from noisy samples
3. SHAP-FOLD learns a reasonable number of non-monotonic rules in the form of default theories
4. SHAP-FOLD is fast and scalable compared to conventional ILP algorithms

This paper makes the following novel contribution: We present a new ILP algorithm capable of learning *non-monotonic* logic programs from local explanations of black-box models provided by SHAP. Our experiments on UCI standard benchmark data sets suggest that SHAP-FOLD outperforms ALEPH [17] in terms of classification evaluation metrics, running time, and providing more concise explanations measured in terms of number of clauses induced.

2 Background

2.1 The FOIL Algorithm

FOIL is a top-down ILP algorithm that follows a *sequential covering* scheme to induce a hypotheses. The FOIL algorithm is summarized in Algorithm 1. This algorithm repeatedly searches for clauses that score best with respect to a subset of positive and negative examples, a current hypothesis and a heuristic called *information gain* (IG).

The inner loop searches for a clause with the highest information gain using a general-to-specific hill-climbing search. To specialize a given clause c, a refinement operator ρ under θ-subsumption [13] is employed. The most general clause

Algorithm 1. Summarizing the FOIL algorithm

```
Input: $target, B, E^+, E^-$
Output: Initialize $H \leftarrow \emptyset$
 1: while ($|E^+| > 0$) do
 2:     $c \leftarrow (target$ :- $true.)$
 3:     while ($|E^-| > 0 \wedge c.length < max\_length$) do
 4:         for all $c' \in \rho(c)$ do
 5:             compute $score(E^+, E^-, H \cup \{c'\}, B)$
 6:         end for
 7:         let $\hat{c}$ be the $c' \in \rho(c)$ with the best score
 8:         $E^- \leftarrow covers(\hat{c}, E^-)$
 9:     end while
10:     add $\hat{c}$ to $H$
11:     $E^+ \leftarrow E^+ \setminus covers(\hat{c}, E^+)$
12: end while
13: return $H$
```

is the following: $p(X_1, ..., X_n) \leftarrow true.$, where the predicate p/n is the predicate being learned and each X_i is a variable. The refinement operator specializes the current clause $h \leftarrow b_1, ...b_n$. This is realized by adding a new literal l to the clause yielding $h \leftarrow b_1, ...b_n, l$. The heuristic based search uses information gain.

2.2 SHAP

SHAP [10] (SHapley Additive exPlanations) is a unified approach with foundations in game theory to explain the output of any machine learning model. Given a dataset and a trained model, the SHAP framework computes a matrix of the shape ($\#samples, \#features$) representing the Shapley value of each feature for each data sample. Each row sums to the difference between the model output for that sample and the expected value of the model output. This difference explains why the model is inclined to predict a specific class outcome.

Example 1. *The UCI heart dataset contains features such as patient's blood pressure, chest pain, thallium test results, number of major vessels blocked, etc. The classification task is to predict whether the subject suffers from heart disease or not. Figure 2 shows how SHAP would explain a model's prediction over a data sample. For this sample, SHAP explains why the model predicts heart disease by returning the top features along with their Shapley values (importance weight). According to SHAP, the model predicts "heart disease" because of the values of "thalium test" and "maximum heart rate achieved" which push the prediction from the base (expected) value of 0.44 towards a positive prediction (heart disease). On the other hand, the feature "chest pain" would have pushed the prediction towards negative (healthy), but it is not strong enough to turn the prediction.*

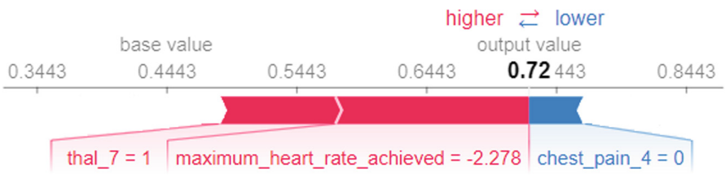

Fig. 2. Shap values for a UCI heart prediction

2.3 High-Utility Itemset Mining

The problem of *High-Utility Itemset Mining* (HUIM) is an extension of an older problem in data mining known as *frequent pattern mining* [1]. Frequent pattern mining is meant to find frequent patterns in transaction databases. A *transaction database* is a set of records (transactions) indicating the items purchased by customers at different times. A *frequent itemset* is a group of items that appear in many transactions. For instance, {noodles, spicy sauce} being a frequent itemset, can be used to take marketing decisions such as co-promoting noodles with spicy sauce. Finding frequent itemsets is a well-studied problem with an efficient algorithm named Apriori [2]. However, in some applications frequency is not always the objective. For example, the pattern {milk,bread} may be highly frequent, but it may yield a low profit. On the other hand, a pattern such as {caviar, champagne} may not be frequent but may yield a high profit. Hence, to find interesting patterns in data, other aspects such as profitability is considered.

Mining high utility itemsets can be viewed as a generalization of the frequent itemset mining where each item in each transaction has a utility (importance) associated with it and the goal is to find itemsets that generate high profit when for instance, they are sold together. The user has to provide a value for a threshold called *minimum utility*. A high utility itemset mining algorithm outputs all the high-utility itemsets with at least *minimum utility* profit. Table 1 shows a transaction database consisting of 5 transactions. Left column shows the transaction Identifier. Middle column contains the items included in each transaction and right column contains each item's respective profit. If the *minimum utility* is set to 25, the result of a high utility itemset mining algorithm is shown in

Table 1. Left: an HUIM Problem Instance. Right: Solution for minutil = 25

Transactions	**Items**	**Profits**
$\mathbf{T_0}$	a b c d e	5 10 1 6 3
$\mathbf{T_1}$	b c d e	8 3 6 3
$\mathbf{T_2}$	a c d	5 1 2
$\mathbf{T_3}$	a c e	10 6 6
$\mathbf{T_4}$	b c e	4 2 3

High Utility Itemsets	
{a, c}: 28	{a, c, e}: 31
{a, b, c, d, e}: 25	{b, c}: 28
{b, c, d}: 34	{b, c, d, e}: 40
{b, c, e}: 37	{b, d}: 30
{b, d, e}: 36	{b, e}: 31
{c, e}: 27	

the right table in Table 1. In order to rank the high utility itemsets, Top-K High Utility Itemset (THUI) mining problem [18] is incorporated in SHAP-FOLD.

3 SHAP-FOLD Algorithm

SHAP-FOLD learns a concept in terms of a default theory [16]. In Logic Programming, default theories are represented using *negation-as-failure* (NAF) semantics [3].

Example 2. *The following default theory "Normally, birds fly except penguins which do not", is represented as:*

```
flies(X) :- bird(X), not ab_bird(X).
ab_bird(X) :- penguin(X).
```

The SHAP-FOLD algorithm adapts the FOIL style sequential covering scheme. Therefore, it iteratively learns single clauses, until all positive examples are covered. To learn one clause, SHAP-FOLD first finds common patterns among positive examples. If the resulted clause (default) covers a significant number of negative examples, SHAP-FOLD swaps the current positive and negative examples and recursively calls the algorithm to learn common patterns in negative examples (exceptions). As shown in Example 2, the exceptions are ruled out using negation-as-failure. Learning exceptions allow our SHAP-FOLD algorithm to distinguish between noisy samples and exceptional cases.

To search for "best" clause, SHAP-FOLD tightly integrates the High Utility Itemset Mining (HUIM) and the SHAP technique. In this novel approach, the SHAP system is employed to find relevant features as well as their importance. To find the "best" clause SHAP-FOLD creates instances of HUIM problem. Each instance, contains a subset of examples represented as a set of "transactions" as shown in Table 1. Each "transaction" contains a subset of feature values along with their corresponding utility (i.e., feature importance). The feature importance $\phi_i \in [0, 1]$ for all i distinct feature values. Therefore, a *high-utility itemset* in any set of "transactions" represents strongest features that would contribute to the classification of a significant number of examples, because, otherwise, that itemset would not have been selected as a high-utility itemset. To find the itemset with highest utility, the HUIM algorithm Top-K [18] is invoked with K set to 1.

SHAP-FOLD takes a target predicate name (G), a tabular dataset (D) with m rows and two different labels $+1$ and -1 for positive examples and negative examples respectively. E^+ and E^- represent these examples in the form of target atoms. It also takes a "transaction" database. Each row of T contains a subset of an example's feature-values ($\boldsymbol{z}_i$) along with their Shapley values ($\boldsymbol{\phi}_i$). This "transaction" database is passed along to create HUIM instance and find the itemset with highest utility every time Top-K algorithm is invoked. The summary of SHAP-FOLD's pseudo-code is shown in Algorithm 2.

In the function FOIL (lines 1–8), *sequential covering* loop to cover positive examples is realized. On every iteration, a default clause (and possibly multiple

Algorithm 2. Summary of SHAP-FOLD Algorithm

```
Input:  $G$: Target Predicate to Learn
        $B$: Background Knowledge
        $D = \{ (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\} : \ y_i \in \{-1, +1\}$
        $E^+ = \{ \boldsymbol{x}_i \mid \boldsymbol{x}_i \in D \wedge y_i = 1\}$ : Positive Examples
        $E^- = \{ \boldsymbol{x}_i \mid \boldsymbol{x}_i \in D \wedge y_i = -1\}$: Negative Examples
        $T = \{ (\boldsymbol{z}_i, \boldsymbol{\phi}_i) \mid \boldsymbol{z}_i \subseteq \boldsymbol{x}_i \wedge \boldsymbol{x}_i \in D \wedge \boldsymbol{\phi}_i$ is $\boldsymbol{z}_i$'s Shapley values $\}$
Output: $D = \{ C_1, ..., C_n\}$                                  ▷ default clauses
        $AB = \{ ab_1, ..., ab_m\}$                      ▷ exceptions/abnormal clauses
 1: function FOIL($E^+, E^-$)
 2:     while ($|E^+| > 0$) do
 3:         $C_{def+exc} \leftarrow$ LEARN_ONE_RULE($E^+$,$E^-$)
 4:         $E^+ \leftarrow E^+ \setminus covers(C_{def+exc}, E^+, B)$
 5:         $D \leftarrow D \cup \{C_{def+exc}\}$
 6:     end while
 7:     return $D, AB$                  ▷ returns sets of defaults and exceptions
 8: end function
 9: function LEARN_ONE_RULE($E^+, E^-$)
10:     - let Item-Set be $\{(f_1, ...f_n), (\phi_1, ..., \phi_n)\} \leftarrow$ TOP-K(K=1,$E^+$,T)
11:     $C_{def} \leftarrow (G$ :- $f_1, ..., f_n)$
12:     $FP \leftarrow covers(C_{def}, E^-)$                  ▷ FP denotes False Positives
13:     if $FP > 0$ then
14:         $C_{def+exc} \leftarrow$ LEARN_EXCEPTIONS($C_{def}, E^-, E^+$)
15:     end if
16:     return $C_{def+exc}$
17: end function
18: function LEARN_EXCEPTIONS($C_{def}, E^+, E^-$)
19:     $\{C_1, ..., C_k\} \leftarrow$ FOIL($E^+, E^-$)          ▷ Recursive Call After Swapping
20:     $ab\_index \leftarrow GENERATE\_UNIQUE\_AB\_INDEX()$
21:     for $i \leftarrow 1$ to $k$ do
22:         $AB \leftarrow AB \cup \{ab_{ab\_index}$ :- $bodyof(C_i)\}$
23:     end for
24:     return $C_{def+exc} \leftarrow (headof(C_{def})$ :- $bodyof(C_{def})$, not$(ab_{ab\_index}))$
25: end function
```

exceptions) - denoted by $C_{def+exc}$ - is learned and added to the hypothesis. Then, the covered examples are removed from the remaining examples. In the function LEARN_ONE_RULE (lines 9–17), Top-K algorithm with $k = 1$ is invoked and a high-utility itemset (i.e., a subset of features-values and their corresponding Shapley values) is retrieved. These subset of features create the default part of a new clause. Next, if the default clause covers false positives, the current positive and negative examples are swapped to learn exceptions. In the function LEARN_EXCEPTIONS (lines 18–25), the algorithm recursively calls itself to learn clauses that would cover exceptional patterns. When the recursive call returns, for all learned clauses, their head is replaced by an abnormality predicate. To manufacture the complete default theory, the abnormality predicate preceded by negation-as-failure (not) is added to the default part. Example 3 shows how SHAP-FOLD learns a concise logic program from an XGBoost trained model.

Example 3. *The "UCI Cars" dataset contains 1728 different cars and their acceptability based on features such as buying price, maintenance cost, trunk size, capacity, number of doors, and safety. SHAP-FOLD generates the following program from a trained XGBoost model:*

```
DEF(1): acceptable(A):- safety(A,high), not ab0(A).
EXCEPTIONS(1): ab0(A):- persons(A,2).
               ab0(A):- maintenance(A,very_high).
DEF(2): acceptable(A):- persons(A,4), safety(A,medium), not ab1(A).
EXCEPTIONS(2): ab1(A):- price(A,very_high), trunk(A,small).
               ab1(A):- price(A,high), maintenance(A,very_high).
DEF(3): acceptable(A):- trunk(A,big), safety(A,medium),
                        persons(A,>5).
```

On first iteration, the clause DEF(1) is generated. Since it covers a significant number of negative examples, E^+ and E^- are swapped and algorithm recursively calls itself. Inside LEARN_EXCEPTIONS, the recursive call returns with EXCEPTIONS(1) clauses. The head predicate `ab0` replaces their head and finally in line 24, the negation of abnormality is appended to the default to create the following default clause: "A car is considered acceptable if its safety is high, unless it only fits two persons or its maintenance cost is high".

4 Experiments

In this section, we present our experiments on UCI standard benchmarks [8].[1] The ALEPH system [17] is used as a baseline. We set ALEPH to use the heuristic enumeration strategy, and the maximum number of branch nodes to be explored in a branch-and-bound search to 500K. We also configured ALEPH to allow up to 50 false examples covered by each clause while each clause is at least 80% accurate. We use precision, recall, accuracy and F_1 score to compare the results.

The SHAP-FOLD requires a statistical model as input to the SHAP technique. While computing the Shapley values is slow, there is a fast and exact implementation called TreeExplainer [9] for ensemble tree models. XGBoost [4] is a powerful ensemble tree model that perfectly works with TreeExplainer. Thus, we trained an XGBoost model for each of the reported experiments in this paper. Table 2 presents the comparison between ALEPH and SHAP-FOLD on classification evaluation of each UCI dataset. The best performer is highlighted with boldface font. In terms of the running time, SHAP-FOLD scales up much better. In case of "King-Rook vs. King-Pawn", while ALEPH discovers 283 clauses in 836 seconds, SHAP-FOLD does much better. It finishes in 8 seconds discovering only 3 clauses that cover the knowledge underlying the model. Similarly, in case of "UCI kidney", SHAP-FOLD finds significantly fewer clauses. Thus,

[1] Full implementation is available at: https://github.com/fxs130430/SHAP_FOLD.

Table 2. Evaluation of SHAP_FOLD on UCI datasets

		Algorithm									
		Aleph					SHAP-FOLD				
Data set	Shape	Precision	Recall	Accuracy	F1	Time (s)	Precision	Recall	Accuracy	F1	Time (s)
Cars	(1728, 6)	0.83	0.63	0.85	0.72	73	**0.84**	**0.94**	**0.93**	**0.89**	**5**
Credit-a	(690, 15)	0.78	0.72	0.78	0.75	180	**0.90**	**0.74**	**0.84**	**0.81**	**7**
Breast-w	(699, 9)	0.92	0.87	0.93	0.89	10	**0.92**	**0.95**	**0.95**	**0.93**	**2**
Kidney	(400, 24)	**0.96**	0.92	0.93	0.94	5	0.93	**0.95**	0.93	0.94	**1**
Voting	(435, 16)	0.97	0.94	0.95	0.95	25	**0.98**	**0.98**	0.95	**0.96**	**1**
Autism	(704, 17)	0.73	0.43	0.79	0.53	476	**0.96**	**0.83**	**0.95**	**0.89**	**2**
Ionosphere	(351, 34)	**0.89**	0.87	0.85	0.88	113	0.87	**0.91**	0.85	**0.89**	**2**
Heart	(270, 13)	0.76	0.75	0.78	0.75	28	0.76	**0.83**	**0.81**	**0.80**	**1**
kr vs. kp	(3196, 36)	0.92	0.99	0.95	0.95	836	0.92	0.99	0.95	0.95	**8**

not only SHAP-FOLD's performance is much better, it discovers more succinct programs. Also, scalability is a major problem in ILP, that our SHAP-FOLD algorithm solves: its execution performance is orders of magnitude better.

SHAP-FOLD almost always achieves a higher Recall score. This suggests that the proper use of *negation-as-failure* leads to better coverage. The absence of negation from ALEPH hypothesis space forces the algorithm to create too specific clauses which leaves many positive examples uncovered. In contrast, our SHAP-FOLD algorithm emphasizes on better coverage via finding high-utility patterns of important features first. If the result turns out to cover too many negative examples to tolerate, by learning exceptions and ruling them out (via the same algorithm applied recursively), SHAP-FOLD maintains the same coverage as it rules out exceptional negative examples.

5 Related Works and Conclusions

A survey of ILP can be found in [12]. In ILP community, researchers have tried to combine statistical methods with ILP techniques. Support Vector ILP [11] uses ILP hypotheses as kernel in dual form of the SVM algorithm. kFOIL [7] learns an incremental kernel for SVM algorithm using a FOIL style specialization. nFOIL [6] integrates the Naive-Bayes algorithm with FOIL. The advantage of our research over all of the above mentioned research work is that, first it is model agnostic, second it is scalable thanks to the fast and scalable HUIM algorithm and SHAP TreeExplainer, third it enjoys the power of *negation-as-failure* which is absent from the above mentioned works.

In this paper, we presented a fast and scalable ILP algorithm to induce default theories from statistical machine learning models. In this novel approach, irrelevant features are filtered out by SHAP, a technique from explainable AI. Then, the problem of searching for "best" clause is reduced to a *High-Utility Itemset Mining* problem. Our experiments on benchmark datasets suggest a significant improvement in terms of the classification evaluation metrics and running time.

References

1. Aggarwal, C.C., Han, J.: Frequent Pattern Mining. Springer, Heidelberg (2014)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of 20th International Conference on Very Large Data Bases, VLDB 1994, pp. 487–499. Morgan Kaufmann Publishers Inc., CA (1994)
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge/New York/Melbourne (2003)
4. Chen, T., Guestrin, C.: Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD, KDD 2016, pp. 785–794 (2016)
5. Gan, W., Lin, J.C., Fournier-Viger, P., Chao, H., Hong, T., Fujita, H.: A survey of incremental high-utility itemset mining. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **8**(2), e1242 (2018)
6. Landwehr, N., Kersting, K., Raedt, L.D.: nFOIL: integrating naïve bayes and FOIL. In: Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, Pittsburgh, Pennsylvania, USA, 9–13 July 2005, pp. 795–800 (2005)
7. Landwehr, N., Passerini, A., Raedt, L.D., Frasconi, P.: kFOIL: learning simple relational kernels. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, MA, USA, 16–20 July 2006, pp. 389–394 (2006)
8. Lichman, M.: UCI, ml repository (2013). http://archive.ics.uci.edu/ml
9. Lundberg, S.M., Erion, G.G., Lee, S.I.: Consistent individualized feature attribution for tree ensembles. arXiv preprint arXiv:1802.03888 (2018)
10. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems, pp. 4765–4774 (2017)
11. Muggleton, S., Lodhi, H., Amini, A., Sternberg, M.J.E.: Support vector inductive logic programming. In: Hoffmann, A., Motoda, H., Scheffer, T. (eds.) DS 2005. LNCS (LNAI), vol. 3735, pp. 163–175. Springer, Heidelberg (2005). https://doi.org/10.1007/11563983_15
12. Muggleton, S., et al.: Ilp turns 20. Mach. Learn. **86**(1), 3–23 (2012)
13. Plotkin, G.D.: A further note on inductive generalization, in machine intelligence, vol. 6, pp. 101–124 (1971)
14. Quinlan, J.R.: Learning logical definitions from relations. Mach. Learn. **5**, 239–266 (1990)
15. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD 2016, pp. 1135–1144 (2016)
16. Shakerin, F., Salazar, E., Gupta, G.: A new algorithm to automate inductive learning of default theories. TPLP **17**(5–6), 1010–1026 (2017)
17. Srinivasan, A.: The Aleph Manual (2001). https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html
18. Tseng, V.S., Wu, C.W., Fournier-Viger, P., Philip, S.Y.: Efficient algorithms for mining top-k high utility itemsets. IEEE Trans. Knowl. Data Eng. **28**(1), 54–67 (2016)