arXiv:2008.03301v1 [cs.AI] 9 Aug 2020

White-box Induction From SVM Models: Explainable AI with Logic Programming

FARHAD SHAKERIN, GOPAL GUPTA
The University of Texas at Dallas, Texas, USA
(*e-mail:* {Farhad.Shakerin, Gopal.Gupta}@utdallas.edu)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

We focus on the problem of inducing logic programs that explain models learned by the support vector machine (SVM) algorithm. The top-down sequential covering inductive logic programming (ILP) algorithms (e.g., FOIL) apply hill climbing search using heuristics from information theory. A major issue with this class of algorithms is getting stuck in a local optima. In our new approach however, the data dependent hill-climbing search is replaced with a model-dependent search where a globally optimal SVM model is trained first, then the algorithm looks into support vectors as the most influential data points in the model, and induces a clause that would cover the support vector and points that are most similar to that support vector. Instead of defining a fixed hypothesis search space, our algorithm makes use of SHAP, an example-specific interpreter in explainable AI, to determine relevant set of features. This approach yields an algorithm that captures SVM model's underlying logic and outperforms other ILP algorithms in terms of the number of induced clauses and classification evaluation metrics. This paper is under consideration for publication in the journal of "Theory and practice of logic programming".

KEYWORDS:Explainable AI, Data Mining, Inductive Logic Programming, Machine Learning

1 Introduction

Dramatic success of machine learning has led to an avalanche of applications of Artificial Intelligence (AI). However, the effectiveness of these systems is limited by the machines' current inability to explain their decisions and actions to human users. That is mainly because the statistical machine learning methods produce models that are complex algebraic solutions to optimization problems such as risk minimization or geometric margin maximization. Lack of intuitive descriptions makes it hard for users to understand and verify the underlying rules that govern the model. Also, these methods cannot produce a justification for a prediction they arrive at for a new data sample. The problem of explaining (or justifying) a model's decision to its human user is referred to as the model interpretability problem. The sub-field is referred to as Explainable AI.

The ILP learning problem is the problem of searching for a set of logic programming clauses that deduce the training examples. ILP is a thriving field and a large number of such clause search algorithms have been devised (Muggleton et al. 2012). The search in these ILP algorithms is performed either top down or bottom-up. A bottom-up approach builds most-specific clauses from the training examples and searches the hypothesis space by using generalization. This approach is not applicable to large-scale datasets, nor it can incorporate *negation-as-failure* (Baral 2003) into the hypotheses. A survey of bottom-up ILP systems and their shortcomings can be found at (Sakama 2005). In contrast, top-down approach starts with the most general clauses and then specializes them. A top-down algorithm guided by heuristics is better suited for large-scale and/or noisy datasets (Zeng et al. 2014).

The FOIL algorithm by Quinlan (Quinlan 1990) is a popular top-down algorithm. FOIL uses heuristics from information theory called *weighted information gain*. The use of a greedy heuristic allows FOIL to run much faster than bottom-up approaches and scale up much better. However, scalability comes at the

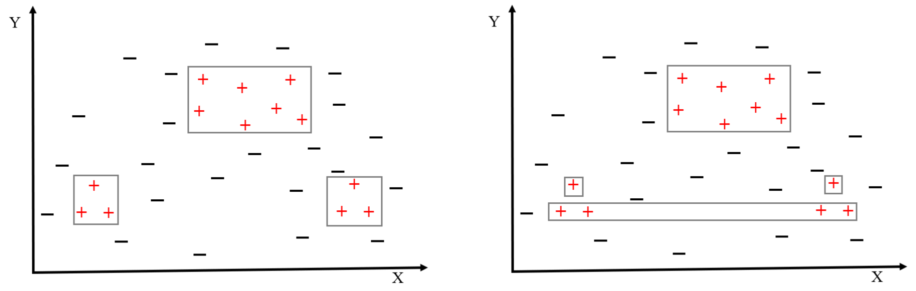

Fig. 1: Optimal sequential covering with 3 Clauses (Left), Sub-Optimal sequential covering with 4 Clauses (Right)

expense of losing accuracy if the algorithm is stuck in a local optima and/or when the number of examples is insufficient. The former is an inherent problem in hill climbing search and the latter is due to the shrinking of examples during clause specialization. Figure 1 demonstrates how the local optima results in discovering sub-optimal rules that do not necessarily coincide with the real sub-concepts they are supposed to capture.

Unlike top-down ILP algorithms, Support Vector Machine (SVM) (Cortes and Vapnik 1995) is a globally optimal learning method that generalizes very well and comes with test error upper-bound in terms of the number of support vectors and size of training input. However, this unique property is overshadowed by the black-box nature of SVM models. Explaining the behavior of black-box models has motivated a long line of research in Rule Induction from SVM models. As we argue in more detail in section 5, all proposed Rule Extraction techniques either treat the model as black-box (Huysmans et al. 2008; Huysmans et al. 2006), or are limited to certain type of kernels (Fung et al. 2005), or are too complex to interpret (Nuez et al. 2002). A survey of existing Rule Extraction techniques can be found in (Diederich 2008).

Our new approach is based on the idea that each data sample is measurably similar/dissimilar to each support vector. Therefore, each support vector represents a subset of data samples. Now, if a set of features discriminate a support vector well, they would discriminate data samples similar to that support vector too. In order to measure the similarity and to pick up the support vector that is most similar to each data sample, we define a quantity based on the kernel value and each support vector's α parameter. To discover the most relevant features, our algorithm incorporates the SHAP technique. SHAP (Lundberg and Lee 2017) is an example specific model interpreter that takes a model and an individual example and returns the contribution of each feature value in model's classification decision.

This paper makes the following novel contribution: it introduces a novel ILP algorithm called SHAP-FOIL that iteratively learns a single clause for the most influential support vector (i.e., the support vector that pulls the highest number of similar data samples), then removes the covered examples including the support vector itself, and repeats this process until all examples/support vectors are covered. Our main claim is that our novel algorithm (SHAP-FOIL) learns a logic program that captures the underlying logic of an SVM model. This logic program serves as a human-understandable explanation of the SVM model. Our experiments using UCI datasets support our claim, as they show that the logic program that SHAP-FOIL learns is human understandable yet has precision, recall, accuracy and F_1-score that is comparable to the original SVM model. Explaining the model as a logic program leads to greater comprehensibility by humans as logic programming has a well-defined declarative and operational semantics. This is in contrast to other rule extraction methods that extract if-then-else rules that tend to have *ad hoc* semantics.

Our work also shows how ILP can leverage the power of statistical machine learning methods to perform the search for induced logic programs significantly better. Our statistical machine learning based ILP method, SHAP-FOIL, performs significantly better than state-of-the-art ILP systems such as ALEPH (Srinivasan 2001). This claim, again, is supported by our experiments done using the UCI datasets, as they show that the logic programs that are learned by SHAP-FOIL have precision, recall, accuracy, and F_1-score that are significantly better than ALEPH. In addition, a logic program that SHAP-FOIL learns is more concise than the program learned by the ALEPH system for the same training set. Thus, SHAP-FOIL also improves comprehensibility (as measured by program conciseness, i.e., program size).

Rest of the paper is organized as follows. Section 2 gives background on various concepts: explainability, ILP, the FOIL algorithm, SVM, and the game theory-based SHAP method. Section 3 presents our novel SHAP-FOIL algorithm. Section 4 presents our experimental results, while Sections 5 discusses related work. Finally, Section 6 presents our conclusions and planned future work.

2 Background

2.1 Explainable AI

Statistical machine learning techniques learn a model from the training data with high precision and accuracy. Once a model is learned, a new, previously unseen input can be given to the model and result obtained. However, these models turn out to be "black boxes" whose internal workings are not understood since they provide no clarity as to how the computed results depend on the input data.

Consider an example where one wants to build a machine learning system to automate the process of approving consumer loans. Training data based on past customers (their income, age, years in job, repayment history, etc.) and the outcome for each (loan approved or declined) will be collected and a machine learning algorithm applied to it. The machine learning algorithm will learn a model to which a new, previously unseen customer's data can be input to determine if this new customer's loan application ought to be approved or not. The model will compute an output that tells us if a customer's loan application should be approved or declined, but it won't tell us or explain to us why. Being able to explain the model's decision is a major problem and its study is referred to as Explainable AI.

We want explainability of the results computed by the model for many reasons some of which we list here:

- It allows users to understand why a particular result was produced and not something else.
- If the result is justifiable, the user will trust the result more.
- It allows users to abide by the law as in many countries machine generated decisions have to come with a justification (e.g, European Union's GDPR).
- It allows a user to judge if the results are correct since if the training data is incorrect or biased, results produced by the model will also be incorrect or biased.

Aside from explainability, we want to ensure that the rules are *comprehensible* by humans, show *fidelity* and are *accurate* (Martens et al. 2008):

- **Comprehensibility**: The extent to which extracted representations are understandable by humans.
- **Fidelity**: The extent to which the extraction representations model the black box.
- **Accuracy**: The ability of extracted representations to make accurate predictions on previously unseen cases.

Comprehensibility is typically measured by the number of rules produced. The assumption is that the fewer the rules, the easier it is for humans to understand the set of rules generated. One can also argue that logic programs are more comprehensible than other formalisms for expressing rules. This is because logic programming has a very well defined declarative and operational semantics.

Note that precision, recall and F_1 score are the typical metrics used to measure fidelity.

2.2 Inductive Logic Programming

Inductive Logic Programming (ILP) (Muggleton 1991) is a subfield of machine learning that learns models in the form of logic programming rules (Horn Clauses) that are comprehensible to humans. This problem is formally defined as:
Given

1. a background knowledge theory B, in the form of an extended logic program, i.e., clauses of the following form:
   $h \leftarrow l_1, ..., l_m,\ \textbf{not}\ l_{m+1}, ...,\ \textbf{not}\ l_n$

where $l_1, ..., l_n$ are positive literals and **not** denotes *negation-as-failure* (NAF) (Baral 2003) and B has no *even cycle* (Gelfond and Kahl 2014)
2. two disjoint sets of ground target predicates E^+, E^- known as positive and negative examples, respectively
3. a hypothesis language of function free predicates L, and a refinement operator ρ under θ-subsumption (Plotkin 1971) that would disallow even cycles.

Find a set of clauses H such that:

- $\forall e \in E^+,\ B \cup H \models e$
- $\forall e \in E^-,\ B \cup H \not\models e$
- $B \wedge H$ is consistent.

While we allow normal logic programs in our definition, we only learn logic programs with no negated literals in our current work (any negated calls in these learned logic program arise due to propositionalisation or one-hot encoding and can be executed using negation as failure as supported in Prolog). Even cycle refers to programs where a recursive call occurs in the scope of even number of negation, for example, the following program has an even cycle.

```
p :- not q.
q :- not p.
```

θ-subsumption (Plotkin 1971) is defined as follows: A clause c theta-subsumes another clause d if and only if there exists a variable substitution θ such that $c\theta \subseteq d$. θ-subsumption determines a partial order of the clause space. Top element of this partial order is the most general clause $p(X_1, ..., X_n) \leftarrow true.$. A refinement operator ρ which moves down this partial order, takes a hypothesis clause c and gives back a set of all syntactically modified version c' of c. In other words, each clause c' is a θ-subsumed specialization of c. For instance, the refinement operator ρ in Algorithm 1 specializes the clause $h \leftarrow b_1, ...b_n.$ by adding a new literal l to the clause yielding $h \leftarrow b_1, ...b_n, l$.

A large number of ILP algorithms have been devised (Muggleton et al. 2012). FOIL (Quinlan 1990) is one such illustrative algorithm for ILP that is well known. FOIL is a top-down ILP algorithm which follows a *sequential covering* scheme to induce a hypothesis. 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).

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 $H$
```

Starting from the most general clause (i.e., $p(X_1, ..., X_n) \leftarrow true.$ where the predicate p/n is the predicate being learned and each X_i is a variable), the inner loop searches for a clause with the highest information gain using a general-to-specific hill-climbing search. To find the "best" clause, first, a refinement operator ρ specializes the current clause $h \leftarrow b_1, ...b_n$. by adding a new literal l to the clause yielding $h \leftarrow b_1, ...b_n, l$. The "best" choice for literal l is decided by a heuristic based search that employs information gain. In FOIL, information gain for a given clause is calculated as follows:

$$IG(L,R) = t\left(log_2 \frac{p_1}{p_1+n_1} - log_2 \frac{p_0}{p_0+n_0}\right) \tag{1}$$

where L is the candidate literal to add to rule R, p_0 (n_0) is the number of positive (negative) examples covered by R respectively, p_1 (n_1) is the number of positive (negative) examples covered by $R+L$ respectively, and t is the number of positive examples that are covered by R and $R+L$ together. The function *covers* in lines 8 and 11, takes a clause c and a set of examples E, and returns another (possibly empty) set of examples that are logically implied by c.

2.3 Support Vector Machines

Given a training dataset of m data points $\vec{x}_i \in \mathrm{I\!R}^n$ and m corresponding labels $y_i \in \{1, -1\}$, the linear support vector machine is defined as the following optimization problem:

$$\min_{\vec{w},b,\xi_i \geq 0} \frac{1}{2}\|\vec{w}\|_2^2 + C\sum_{i=1}^{m} \xi_i \tag{2}$$

such that: $y_i(\vec{w}.\vec{x}_i - b) + \xi_i \geq 1$ where ξ_i is the slack error to potentially allow some points to be misclassified, $\vec{w}$ is the perpendicular vector to the separating hyper-plane, b is the offset of that hyper-plane, C is a hyperparameter and determines the degree to which misclassification is allowed to avoid over-fitting.

An equivalent yet more efficient form of SVM problem known as dual formulation is defined as follows:

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{m} y_i y_j \vec{x}_i \vec{x}_j \alpha_i \alpha_j + \sum_{i=1}^{m} \alpha_i \tag{3}$$

such that $\sum_{i=1}^{m} \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$
The dot product in dual formulation can be replaced with any kernel function (i.e., a function that maps data into higher dimensions). For non-linearly separable data, kernels map the data into a higher dimensional feature space where data becomes separable. Equation (3) is a special case for the following general dual formulation:

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{m} y_i y_j K(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j + \sum_{i=1}^{m} \alpha_i \tag{4}$$

such that $\sum_{i=1}^{m} \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$
After solving the above problem using quadratic programming, the α_i and b are used to classify a new data sample $\vec{x}$ as follows:

$$f(x) = sign\left[\sum_{i=1}^{m} \alpha_i y_i K(\vec{x}_i, \vec{x}) + b\right] \tag{5}$$

It turns out that α is a sparse vector and only few α_i come back with non-zero values from the quadratic solver package. They are called the support vectors and as Equation (5) suggests, proportionate to their respective α_i value, support vectors are the only influential data points in classification decision of any new data sample.

While the kernel function is meant to map data points into a higher dimension efficiently, one can also interpret the kernel value $K(\vec{x}_i, \vec{x}_j)$ as a similarity measure between points $\vec{x}_i$ and $\vec{x}_j$. For instance, in case of the Gaussian radial basis kernel (rbf):

$$K(\vec{x}_i, \vec{x}_j) = e^{-\gamma\|\vec{x}_i - \vec{x}_j\|_2^2} \tag{6}$$

where γ is a hyper parameter. If $\vec{x}_i$ and $\vec{x}_j$ are similar, the kernel value would be close to 1. Otherwise, it would be close to 0. This can be naturally used to quantify similarity. However, it should be noted from Equation (5) that the magnitude of α_i also contributes to the influence and similarity. Therefore, we define

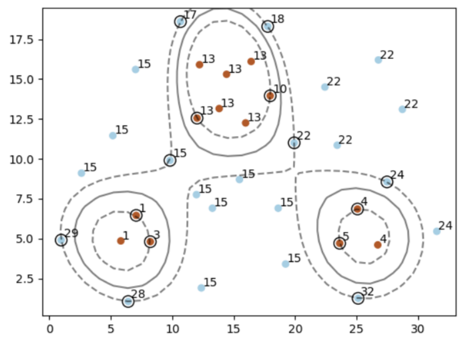

Fig. 2: Annotating Data Points in a 2D dataset With Most Similar Support Vector

the similarity of data point $\vec{x}$ and the i^{th} support vector $\vec{x}_i$ as follows:

$$sim_i(\vec{x}) = \alpha_i y_i K(\vec{x}_i, \vec{x}) \tag{7}$$

For any new data sample $\vec{x}$ the support vector with highest *sim* value is the one that contributes most to the prediction of $\vec{x}$. Figure 2 demonstrates an SVM model with rbf kernel trained on the same dataset from Figure 1. In this figure, support vectors are the dots located on the dashed lines. They are labeled with an integer identifier. Every other data point is annotated with the identifier of the most similar support vector calculated using Equation (7). Since the similarity is measured with respect to the concept being learned, the most similar support vector does not necessarily accord with the Euclidean distance.

2.4 SHAP

SHAP (Lundberg and Lee 2017) (SHapley Additive exPlanations) is a unified approach with foundations in game theory to explain the output of any machine learning model in terms of its features' contributions. To compute each feature i's contribution, SHAP requires retraining the model on all feature subsets $S \subseteq F$, where F is the set of all features. For any feature i, a model $f_{S\cup\{i\}}$ is trained with the feature i present, and another model f_S is trained with feature i eliminated. Then, the difference between predictions is computed as follows: $f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)$, where x_S represents sample's feature values in S. Since the effect of withholding a feature depends on other features in the model, the above differences are computed for all possible subsets of $S \subseteq F \setminus \{i\}$ and their average taken. The weighted average of all possible differences (a.k.a Shapley value) is used as feature importance. Equation (8) shows how Shapley value associated with each feature value is computed:

$$\phi_i = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S) \right] \tag{8}$$

Given a dataset and a trained model, SHAP outputs a matrix with 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 on predicting 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 3 shows how SHAP would explain a model's prediction over a data sample.

For this individual, 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.

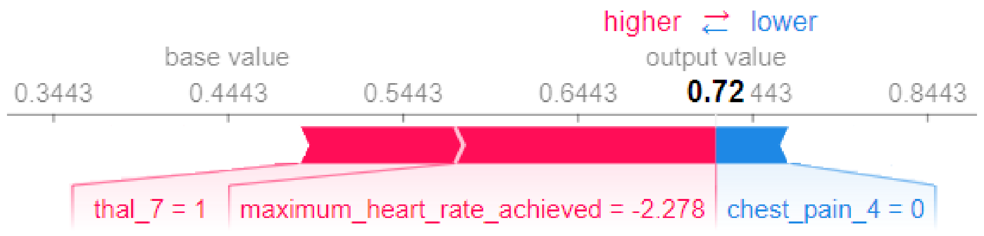

Fig. 3: Shap Values for A UCI Heart Prediction

The categorical features should be *binarized* before an SVM model can be trained. Binarization (aka one-hot encoding) is the process of transforming each categorical feature with domain of cardinality n, into n new binary predicates (features). In Example 1, chest pain level is a categorical feature with 4 different values in the set $\{1,2,3,4\}$. Type 4 chest pain indicates asymptomatic pain and is a serious indication of a heart condition. In this case, binarization results in 4 different predicates. The "thalium test" is also a categorical feature with outcomes in the set $\{3,6,7\}$. Any outcome other than 3, indicates a defect (6 for fixed and reversible for 7).

In case of Example 1, SHAP determines that the feature "thal_7" with outcome of 1 (True), pushes the prediction towards heart disease. To reflect this fact in our SHAP-FOIL ILP algorithm (described next), for any person X, the predicate `thal(X,7)` is introduced. Also, SHAP indicates that the binary feature "chest_pain_4" with value 0 (False), pushes the prediction towards healthy. In our SHAP-FOIL algorithm this is represented by *negation-as-failure* (Baral 2003) as `not chest_pain(X,4)`. Note that, at present, negation-as-failure is used only to succinctly represent such one-hot encoded predicates.

3 The SHAP-FOIL Algorithm

In this section we introduce SHAP-FOIL, an algorithm capable of learning logic programs based on the global behavior of an SVM model.

There are two major issues with the *sequential covering* algorithms such as FOIL: 1) As number of examples decreases during specialization loop, probability of introducing an irrelevant predicate that accidentally splits a particular set of examples increases. 2) The greedy nature of *hill-climbing* search in clause specialization sometimes results in introduction of wrong predicates that would cover more examples at a certain moment, but eventually leads to inducing a clause that does not perfectly represent sub-concepts as shown in Example 1. This is known as Local Optima problem in *hill-climbing* search. Determining the sub-concepts requires a global view which could only happen via a global optimization process such as an SVM model. However, finding the best separating hyperplane in a higher dimension does not explain the contributing features in any classification decision made by the model.

SHAP is able to quantitatively explain the features that would push the model towards predicting a specific outcome. In particular, for each support vector, SHAP determines a subset of feature value pairs that would make the model arrive at a certain decision. It turns out that just by having the Shapley values of support vectors, our algorithm can learn the global underlying behavior of SVM model. This is mathematically justified as follows: From Equation (5), every new data sample is interpreted in terms of similarity to support vectors. The internal points are not relevant (because their corresponding α_i parameter is 0). Among support vectors, only the ones that are closely "similar" to the given point are relevant (because, for dissimilar support vectors the kernel value of Equation (6) is close to 0).

The intuition behind SHAP-FOIL algorithm is as follows: If a subset of feature-values explains the decision on a particular support vector, it explains the decision on data points that are "similar" to that support vector too. Similarity is measured using Equation (7). In the context of *sequential covering* scheme, SHAP-FOIL would find the support vector that pulls the greatest number of data points in terms of Equation

(7). Next, it would specialize a clause by introducing predicates that are determined by SHAP for that support vector. Then, the algorithm removes the data points that are covered by that rule. It also removes the support vector. This process is repeated for the remaining data points. Since there are only finite number of support vectors, the algorithm is guaranteed to terminate.

Unlike most ILP algorithms, SHAP-FOIL does not require discretization of numerical features in advance. During the specialization of a clause, if a numeric feature happens to have the highest Shapley number for a support vector, a real arithmetic constraint is introduced by the algorithm. The end-points of this interval is determined by looking into the respective values of all data points that are most similar to that support vector.

Algorithm 2 Summarizing the SHAP-FOIL algorithm

```
Input: $D = \{(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)\}$
Input: $SHAP\ matrix\ (m, \#features), \Theta$
Input: $S = \{sv \mid sv\ is\ a\ support\ vector\ and\ sv\ is\ TP\}$
Output: $Hypothesis\ H = \{\}$
 1: function SHAP-FOIL($S, D$)
 2:     while ($S \neq \emptyset$) do
 3:         $sim\_map$ = ANNOTATE_SAMPLES($S$,$D$)
 4:         $sv = \text{argmax}_{s \in S}\, len(sim\_map[s])$
 5:         $c \leftarrow (target$ :- $true.)$
 6:         $\hat{c} \leftarrow$ Specialize $c$ using SHAP[sv]
 7:         if $\hat{c}$'s accuracy on D $\geq \Theta$ then
 8:             $H \leftarrow H \cup \{\hat{c}\}$
 9:             $D \leftarrow D - \{\vec{x}_i \mid \vec{x}_i \in D \wedge \hat{c} \models y_i\}$
10:         end if
11:         $S \leftarrow S - \{sv\}$
12:     end while
13:     return $H$
14: end function
15:
16: function ANNOTATE_SAMPLES($S, D$)
17:     Let $sim\_map$ be a map of type : $S \mapsto List$
18:     for each $sv \in S$ do
19:         $sim\_map[sv] = [\,]$
20:     end for
21:     for each $\vec{x}_i \in D$ do
22:         $sv = \text{argmax}_{s \in S}\, sim(s, \vec{x}_i)$
23:         $sim\_map[sv].append(\vec{x}_i)$
24:     end for
25:     return $sim\_map$
26: end function
```

Algorithm 2 summarizes the SHAP-FOIL algorithm. The algorithm inputs are as follows: 1) Set D of cardinality m representing m training data points. 2) SHAP matrix of a trained SVM model on D. 3) A threshold Θ to determine minimum acceptable accuracy of each induced clause. 4) Set *SV* that are True Positive (TP) support vectors (i.e., support vectors with label 1, also predicted 1 by the SVM model). The model outputs a hypothesis comprising a set of induced clauses. The While loop in line 2, iterates until all support vectors are considered. (termination condition). In line 3, by calling the function ANNOTATE_SAMPLES, all remaining data-points are annotated with a support vector among the remaining support vectors. This support vector must have the highest similarity value to that data point. In line 4, the algorithm chooses *sv*, the support vector which pulled the greatest number of data points from annotation. This allows to discover the more inclusive rules first. In lines 5 and 6, similar to the FOIL algorithm, specialization from the most general clause (i.e., `target :- true.`) is conducted. To specialize a clause, predicates determined by the Shapley value of *sv* are added to the body in the order of their Shapley value magnitude. To add numeric features, our algorithm creates an interval by finding the smallest and largest values in the list of data samples associated with *sv*. The specialized clause is named $\hat{c}$. In line 7, the accuracy of $\hat{c}$ is tested against a threshold Θ. If it is higher than Θ, $\hat{c}$ is added to the current hypothesis H in line 8. In line 9, similar to FOIL, the set of data points covered by $\hat{c}$ are removed from D (*sequential covering*). If $\hat{c}$ achieves lower accuracy than Θ, it is discarded. Regardless of the case, in line 11, *sv* is removed from the set of support vectors. This serves two purposes: 1) It guarantees the termination. 2) More importantly, if in some iteration, a support vector pulls greatest number of similar points but it does not yield an above Θ accurate clause, to make progress possible, this support vector will be removed from consideration. We will clarify this more in Example 3.

Example 2
Figure 1 illustrates the local optima issue of FOIL. In Figure 2, an SVM model is shown for the same dataset with two features f1 and f2 and two classes of red and blue. The SHAP-FOIL algorithm learns the following logic program on this dataset for the class red. While there are six support vectors, the set of red example data points were found most similar to only three support vectors. As a a result, only three rules were generated for "red" concept, shown below. The 'blue' concept can be learned similarly.

```
red(X):- f1(X,F1), 12.02 =< F1 <= 17.97,
         f2(X,F2), 12.25 =< F2 <= 16.1 .
red(X):- f1(X,F1), 5.82 =< F1 <= 8.22,
         f2(X,F2), 4.8 =< F2 <= 6.45 .
red(X):- f1(X,F1), 23.62 =< F1 <= 26.72,
         f2(X,F2), 4.6 =< F2 <= 6.85 .
```

To handle numeric features, our algorithm creates an interval by finding the smallest and largest values in the list of data samples associated with a support vector *sv*. This approach sometimes results in too coarse-grained intervals that cover too many False Positives (FP) to tolerate. As explained earlier, to handle this case, SHAP-FOIL removes *sv* and tries to break the region into smaller sub-regions. Each smaller region is then covered by other support vectors. We illustrate this with an example.

Example 3
Consider a dataset of two features on which an SVM model is trained (Figure 4). At iteration 1, after annotating the data samples, the support vector 3, pulls the majority of data samples. According to SHAP, both numeric features contribute to the classification decision. However, after creating the intervals for both features, the clause shown as a green box, ends-up covering significant number of False Positives. This is shown in Figure 4. Therefore, the clause is discarded and the support vector 3, is removed from consideration. As shown in Figure 5, on iteration 2, the support vector 8 pulls the highest number of data samples. It yields an accurate clause. Thus, it is added to the hypothesis. On iteration 3, the support vector 2 pulls the rest of data samples and once again, it results in an accurate clause.

4 Experimental Results

In this section, we present our experiments on UCI standard benchmarks (Dua and Graff 2017). We compare the performance of the logic program output by SHAP-FOIL with that of the original SVM model and the ALEPH ILP system (Srinivasan 2001). ALEPH is a state-of-the-art ILP system that has been widely used in prior work. To find a rule, ALEPH starts by building the most specific clause, which is called the "bottom clause", that entails a seed example. Then, it uses a branch-and-bound algorithm to perform a general-to-specific heuristic search for a subset of literals from the bottom clause to form a more general rule. We set ALEPH to use the heuristic enumeration strategy, and the maximum number of branch nodes to be explored

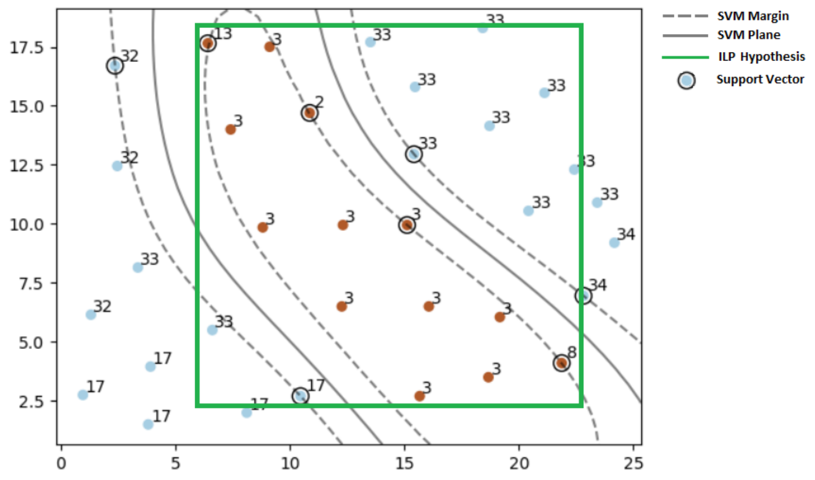

Fig. 4: Iteration #1 of Example 3

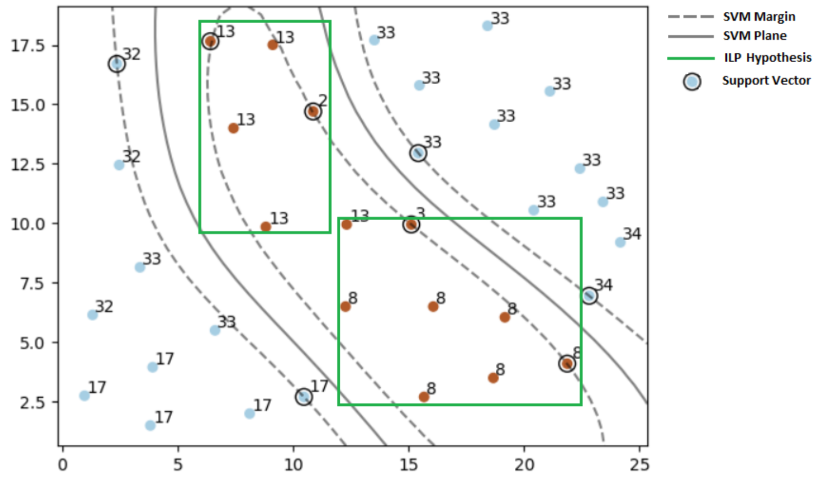

Fig. 5: Iteration #2, #3 of Example 3

in a branch-and-bound search to 500K. We use the standard metrics including precision, recall, accuracy and F_1 score to measure the quality of the results.

Table 1 presents the comparison between ALEPH and SHAP-FOIL on classification evaluation of each UCI dataset. It also shows the results for the SVM method. The best performer is highlighted with boldface font. With the exception of congressional voting dataset where the SVM performance is lower than ALEPH, the SHAP-FOIL algorithm always achieves higher score compared to ALEPH. Our experiments illustrate two important points:

1. A logic program induced by SHAP-FOIL not only explains the black-box SVM model, it performs well wrt all classification evaluation metrics, thus exhibiting high fidelity (quantitatively measured by F_1-score).
2. Our ILP algorithm in which learning of clauses is guided by a model-dependent search is superior in performance to traditional ILP algorithms such as ALEPH. This insight can be adopted by other ILP algorithms to make them better.

		Algorithm											
		SVM				Aleph				SHAP-FOIL			
Data Set	Kernel	Prec.	Recall	Acc.	F_1	Prec.	Recall	Acc.	F_1	Prec.	Recall	Acc.	F_1
credit-j	rbf	0.84	0.84	0.84	0.84	0.78	0.72	0.78	0.75	**0.83**	**0.76**	**0.83**	**0.80**
breast-w	Poly	0.97	0.96	0.96	0.96	0.92	0.87	0.93	0.89	**0.97**	**0.89**	**0.95**	**0.93**
ecoli	rbf	0.96	0.96	0.96	0.96	0.85	0.75	0.84	0.80	**0.86**	**0.94**	**0.89**	**0.90**
kidney	poly	0.99	0.99	0.99	0.99	0.96	0.92	0.93	0.94	**0.97**	**0.97**	**0.97**	**0.97**
voting	rbf	0.95	0.94	0.94	0.94	**0.97**	**0.94**	**0.95**	**0.95**	0.92	0.94	0.91	0.93
autism	rbf	1.00	1.00	1.00	1.00	0.73	0.43	0.79	0.53	**0.94**	**0.86**	**0.94**	**0.88**
ionosphere	rbf	0.95	0.95	0.94	0.95	0.89	0.87	0.85	0.88	**0.92**	**0.90**	**0.90**	**0.91**
heart	poly	0.81	0.80	0.80	0.80	0.76	0.75	0.78	0.75	**0.90**	**0.86**	**0.90**	**0.88**

Table 1: Evaluation of SHAP-FOIL on UCI Datasets

The *sequential-covering* based algorithms—including ALEPH and FOIL—tend to learn too many rules in presence of noisy data. Both algorithms induce more accurate clauses at the expense of covering fewer examples by each clause. In our SHAP-FOIL algorithm, specialization is stopped once the purity of a clause reaches the threshold Θ while the maximum coverage is guaranteed by SHAP because the specialization is performed in the order of feature's Shapley number. For instance, while ALEPH discovers 15 clauses for UCI heart, the following logic program comprised of only 6 clauses is induced by the SHAP-FOIL algorithm:

```
(1) heart_disease(X) :- thallium_test(X,7),
                        chest_pain(X,4),
                        exercise_induced_angina(X).

(2) heart_disease(X) :- maximum_heart_rate_achieved(X,F1),
                        106 =< F1, F1 =< 154,
                        not major_vessels(X,0),
                        oldpeak(X,F2),
                        1 =< F2, F2 =< 4.
(3) heart_disease(X) :- not major_vessels(X,0),
                        thallium_test(X,7),
                        chest_pain(X,4).
(4) heart_disease(X) :- thallium_test(X,7),
                        age(X,F1),
                        35 =< F1, F1 =< 52,
                        chest_pain(X,4).
(5) heart_disease(X) :- maximum_heart_rate_achieved(X,F1),
                        120 =< F1, F1 =< 147,
                        exercise_induced_angina(X),
                        chest_pain(X,4).
(6) heart_disease(X) :- not major_vessels(X,0),
                        chest_pain(X,4),
                        male(X).
```

The induced program can be understood as follows: In clause (1), `thallium_test(X,7)` indicates a *thallium test* with reversible defect, while `chest_pain(X,4)` indicates an asymptomatic type of chest pain. According to clause (1), these two conditions, conjoined with angina revealed in an exercise test indicate the existence of heart disease. In clause (2), if maximum heart rate is achieved and during exercise test it falls in the discovered range 106-154, ST depression induced by exercise relative to rest falls in the range 1-4, and there are signs of blockage in major vessels (indicated by *negation-as-failure*), then the combination means heart disease. The rest of the clauses are read similarly.

As stated earlier, our SHAP-FOIL algorithm not only does better than ALEPH in classification evaluation measures, it also produces much smaller number of rules. In many cases, our SHAP-FOIL algorithm produces an order of magnitude fewer rules than the ALEPH system. Obviously, smaller number of rules are more readily understood by users. They can be manually revised by the user much more easily as well (to better capture the learned knowledge) based on user's background knowledge about the problem. Thus, logic programs discovered by our SHAP-FOIL algorithm are more comprehensible as they are shorter. Table 2 shows the size of the program discovered by ALEPH vs. SHAP-FOIL.

Dataset	credit-J	breast-w	ecoli	kidney	voting	autism	ionosphere	sonar	heart
ALEPH	30	20	13	10	7	11	13	10	13
SHAP-FOIL	22	5	8	5	5	6	14	6	6

Table 2: Number of lines in logic programs induced by Aleph and SHAP-FOIL

It should be noted that the comprehensibility of the program generated by SHAP-FOIL is superior to those generated by various rule-extraction methods

5 Related Works

As noted earlier, all proposed Rule Extraction techniques either treat the model as black-box (Huysmans et al. 2008; Huysmans et al. 2006), or are limited to certain type of kernels (Fung et al. 2005), or are too complex to interpret (Nuez et al. 2002). A survey of existing Rule Extraction techniques can be found in (Diederich 2008). In contrast, our SHAP-FOIL method produces a succinct logic program that has an elegant declarative as well as operational semantics resulting in better comprehensibility. Most rule extraction methods adopt an ad hoc if-then-else notation for representing the rules they output.

Rule extraction from statistical Machine Learning models has been a long-standing goal of the community. The rule extraction algorithms from machine learning models are classified into two categories: 1) Pedagogical (i.e., learning symbolic rules from black-box classifiers without opening them) 2) Decompositional (i.e., to open the classifier and look into the internals). TREPAN (Craven and Shavlik 1995) is a successful pedagogical algorithm that learns decision trees from neural networks. Minerva (Huysmans et al. 2008) and Iter (Huysmans et al. 2006) are pedagogical approaches to extract rules from SVM models. There is also a broader pedagogical approach to rule extraction where an SVM model is trained first, then the entire training data is re-labeled using the predictions of the SVM model, and finally a rule learning method (e.g., C4.5, ID3, CART etc.) is applied. It should be noted that this approach suffers from the very issue of local optima discussed in Figure 1.

SVM+Prototypes (Nuez et al. 2002) is a decompositional rule extraction algorithm that makes use of KMeans clustering to extract rules from SVM classifiers by focusing on support vectors. The main drawback of this approach is that all extracted rules contain all possible input variables in its conditions, making the approach too complex to interpret for large input dimensions. Fung (Fung et al. 2005) is another decompositional SVM rule extraction technique to extract propositional rules which is limited to linear kernels. The advantage of our SHAP-FOIL algorithm is that it can handle all kernels and the induced hypotheses are expressed in terms of the original features.

6 Conclusions and Future Work

In this paper we presented an algorithm to open up the Support Vector Machine (SVM) models and induce logic programs by looking at the support vectors as the most influential points in making classification decision. We make use of SHAP (Lundberg and Lee 2017) to find the most relevant features that would contribute to the support vector's prediction. We showed that finding data points "similar" to influential support vectors allows same feature values to induce a hypothesis that correctly predicts internal data points. The resulting algorithm significantly outperforms state-of-the-art ILP system ALEPH in terms of the classification evaluation measures while it does not suffer from the local optima issue as explained earlier.

There are number of directions for future work: (i) Equation (7) can measure dissimilarity too. Dissimilarity is usually expressed as *negation-as-failure* in ILP (Shakerin et al. 2017). We plan to incorporate dissimilarity to support vectors of the opposite class as a way to reduce the number of induced clauses as well as increase the accuracy of each clause. In such a case, we will be learning answer set programs (Gelfond and Kahl 2014). Learning answer set programs has an added advantage of being able to distinguish between exceptions and noise (Shakerin et al. 2017; Shakerin and Gupta 2020). (ii) In sparse regions of training data, specialization is usually stopped too early. If the algorithm can introduce synthesized data and have the SVM model label it, the SHAP-FOIL algorithm can continue specialization further and achieve better test accuracy. (iii) Our work thus far has focused on classification problems. We plan to generalize and extend our work to induce logic programs from regression models in the future. This will be achieved by discretization of continuous spaces. It should be noted that we have applied our technique to gradient-boosting models as well (Shakerin and Gupta 2020) with excellent results.

Acknowledgment

Authors acknowledge support from NSF grants IIS 1718945, IIS 1910131, IIP 1916206 and DARPA grant HR001119S0057-ARCOS-FP-036.

References

Charu C. Aggarwal and Jiawei (editors) Han. 2014. *Frequent Pattern Mining*. Springer International Publishing.

Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487–499.

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

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD (KDD '16)*. 785–794.

Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1023/A:1022627411411

Mark W. Craven and Jude W. Shavlik. 1995. Extracting Tree-structured Representations of Trained Networks. In *Proceedings of the 8th International Conference on Neural Information Processing Systems (NIPS'95)*. MIT Press, Cambridge, MA, USA, 24–30.

Joachim Diederich. 2008. *Rule Extraction from Support Vector Machines: An Introduction*. Springer Berlin Heidelberg, Berlin, Heidelberg, 3–31.

Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

Philippe Fournier-Viger, Jerry Chun-Wei Lin, Tin Truong-Chi, and Roger Nkambou. 2019. A Survey of High Utility Itemset Mining. In *High-Utility Pattern Mining: Theory, Algorithms and Applications*. Springer International Publishing, 1–45. https://doi.org/10.1007/978-3-030-04921-8_1

Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S. Tseng. 2014. FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning. In *Foundations of Intelligent Systems*, Troels Andreasen, Henning Christiansen, Juan-Carlos Cubero, and Zbigniew W. Raś (Eds.). Springer International Publishing, 83–92.

Glenn Fung, Sathyakama Sandilya, and R. Bharat Rao. 2005. Rule Extraction from Linear Support Vector Machines. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05)*. ACM, New York, NY, USA, 32–40.

Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Tzung-Pei Hong, and Hamido Fujita. 2018. A survey of incremental high-utility itemset mining. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 8, 2 (2018).

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

David Gunning. 2015. Explainable Artificial Intelligence (XAI), https://www.darpa.mil/program/explainable-artificial-intelligence. Retrieved June 2018.

Johan Huysmans, Bart Baesens, and Jan Vanthienen. 2006. ITER: an algorithm for predictive regression rule extraction. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer, Krakow, Poland, 270–279.

Johan Huysmans, Rudy Setiono, Bart Baesens, and Jan Vanthienen. 2008. Minerva: Sequential covering for rule extraction. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38, 2 (2008), 299–309.

Niels Landwehr, Kristian Kersting, and Luc De Raedt. 2005. nFOIL: Integrating Naïve Bayes and FOIL. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. 795–800.

Niels Landwehr, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. 2006. kFOIL: Learning Simple Relational Kernels. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. 389–394.

Scott M Lundberg, Gabriel G Erion, and Su-In Lee. 2018. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888* (2018).

Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., Long Beach, CA, 4765–4774.

David Martens, Johan Huysmans, Rudy Setiono, Jan Vanthienen, and Bart Baesens. 2008. Rule Extraction from Support Vector Machines: An Overview of Issues and Application in Credit Scoring. In *Rule Extraction from Support Vector Machines*, Joachim Diederich (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–63.

Stephen Muggleton. 1991. Inductive Logic Programming. *New Gen. Comput.* 8, 4 (Feb. 1991), 295–318.

Stephen Muggleton, Luc de Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. 2012. ILP Turns 20. *Mach. Learn.* 86, 1 (Jan. 2012), 3–23.

Stephen Muggleton, Huma Lodhi, Ata Amini, and Michael J. E. Sternberg. 2005. Support Vector Inductive Logic Programming. In *Discovery Science*, Achim Hoffmann, Hiroshi Motoda, and Tobias Scheffer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg.

Haydemar Nuez, Cecilio Angulo, and Andreu Catal. 2002. Rule extraction from SVM. In *Proc. The European Symposium on Artificial Neural Networks*. Bruges, Belgium, 107–112.

G. D. Plotkin. 1971. A further note on inductive generalization. In *Machine Intelligence* (1971), Vol. 6. Edinburgh University Press, pages 101–124.

J. Ross Quinlan. 1990. Learning Logical Definitions from Relations. *Machine Learning* 5 (1990), 239–266.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD 2016*. 1135–1144.

F. Riguzzi. 2016. ALEPH in SWI-Prolog. https://github.com/friguzzi/aleph

Chiaki Sakama. 2005. Induction from answer sets in nonmonotonic logic programs. *ACM Trans. Comput. Log.* 6, 2 (2005), 203–231.

Farhad Shakerin and Gopal Gupta. 2020. Whitebox Induction of Default Rules Using High-Utility Itemset Mining. In *Practical Aspects of Declarative Languages - 22nd International Symposium, PADL 2020, New Orleans, LA, USA, January 20-21, 2020, Proceedings (Lecture Notes in Computer Science)*, Ekaterina Komendantskaya and Yanhong Annie Liu (Eds.), Vol. 12007. Springer, 168–176. https://doi.org/10.1007/978-3-030-39197-3_11

Farhad Shakerin, Elmer Salazar, and Gopal Gupta. 2017. A new algorithm to automate inductive learning of default theories. *TPLP* 17, 5-6 (2017), 1010–1026.

Ashwin Srinivasan. 2001. The Aleph Manual. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

Vincent S Tseng, Cheng-Wei Wu, Philippe Fournier-Viger, and S Yu Philip. 2016. Efficient algorithms for mining top-k high utility itemsets. *IEEE Trans. on Know. and Data Engg.* 28, 1 (2016), 54–67.

Jan Wielemaker. 2020. The SWI-Prolog System. http://www.swi-prolog.org/

Qiang Zeng, Jignesh M. Patel, and David Page. 2014. QuickFOIL: Scalable Inductive Logic Programming. *Proc. VLDB Endow.* 8, 3 (Nov. 2014), 197–208.