Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 342 (2019) 39–55

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

An Adaptive Algorithm for Rule Learning: Case Study and Preliminary Results

Renata Luiza Stange[a,b,1,2] Paulo Roberto Massa Cereda[b,3]
João José Neto[b,4]

[a] *Federal University of Technology*
Guarapuava, Paraná, Brazil

[b] *University of São Paulo,*
São Paulo, São Paulo, Brazil

Abstract

Problem partitioning strategies such as sequential covering are commonly used in rule learning algorithms, such that the task of finding a complete rule base is reduced to a sequence of subproblems. In this scenario, each solution to a subproblem consists of adding a single rule to the entire set. In this paper, we propose an alternative for rule learning based on the use of an adaptive formalism whose behavior is determined by a dynamic set of rules. Preliminary results yield a compact yet significantly comprehensible rule set, as well as an efficient model representation, from raw data using adaptive techniques. To this end, we include further discussions regarding features and enhancements to be contemplated in future works.

Keywords: artificial intelligence, machine learning, rule learning, adaptivity, problem partitioning

1 Introduction

Machine learning is concerned about the construction of computer programs that automatically improve with experience [13]. Similarly, pattern recognition is interested in the automatic discovery of regularities in data through computer algorithms and the later application of such regularities as decision making actions, such as different data categorizations [3]. One of the well-known classification approaches in machine learning consists of extracting rules. These algorithms generally produce IF-THEN classifiers, with a predictive performance comparable to other traditional classification approaches, such as decision trees and associative classification [22].

[1] E-mail: rlgomes@utfpr.edu.br
[2] E-mail: rlstange@usp.br
[3] E-mail: paulo.cereda@usp.br
[4] E-mail: jjneto@usp.br

https://doi.org/10.1016/j.entcs.2019.04.004

A inductor method for rule classification applies an iterative process that covers a subset of training examples and then remove all examples covered by the rule from the training set. This process is repeated until there are no examples left to cover. The final rule set is the collection of rules learned at each iteration [9]. PRISM is one of the rule induction techniques which was developed in [4] and slightly enhanced by others, i.e. [8] and [19]. This algorithm employs separate-and-conquer strategy in knowledge discovery in which PRISM generates rules according to the class labels in the training dataset.

Adaptive technology refers to the use of techniques and devices which are expected to react to given inputs by autonomously modifying their own behavior [14]. In a broad sense, computers learn when there is a behavioral change in order to better perform a specific task. Inspired by previous works on these areas [20,21], we demonstrate how learning systems can be dynamically adjusted by using adaptive techniques. A direct advantage on incorporating adaptivity in learning methods consists on covering a fundamental aspect of learning itself: the dynamic adaptation of a rule set based on interactions with the environment [15]. Additionally, use of adaptive technology tends to be more expressive than traditional methods [14].

This paper presents a hybrid approach to extracting rules from data using adaptive concepts and supervised learning techniques, such as obtaining IF-THEN rules from data. The approach is based on sequential covering strategies which involve problem decomposition: the task of finding a complete rule base is reduced to a sequence of subproblems in which the solution to each subproblem is a single rule. The global solution gathers all partial solutions [9]. Additionally, the self-modification feature of adaptive rule-driven devices allows an iterative knowledge inspection, adding rules that improve predictions on the complete rule base and also replacing one or more rules in order to simplify the knowledge representation. Observe that algorithms AQ [12] and CN2 [5] implement this strategy.

Adaptive technology has been successfully used in pattern recognition and machine learning. Pistori and Neto [16] propose a decision tree induction algorithm using adaptive techniques, combining syntactic and statistical strategies. In [17], Pistori presents an adaptive automaton as device for an automatic recognition process of sign language. Adaptive automata are also reported to be used in syntactic pattern recognition of shapes [6] and in construction of hybrid maps for robot navigation [11]. Other applications of adaptive techniques include skin cancer recognition [10] and optical character recognition [7].

2 Theoretical framework

This section provides the theoretical framework needed for our proposal, covering the classification problem itself, rule-based systems and rule-driven adaptive devices.

2.1 Rule-based systems

Decision rules are widely used to represent knowledge obtained from data [2]. An IF-THEN rule has a simple construction; for instance, if an certain object swims and

has scales, then such object is a fish. Fig. 1 shows the structure of an rule. A rule-based system can be learned through a strategy known as *divide and conquer*. Rule-based methods are fundamental to expert systems in artificial intelligence, where classes can be characterized by general relationships among entities. Here we shall focus on a broad class of IF-THEN rules for representing and learning such relationships. The general form of a IF-THEN rule is $\mathtt{P} \rightarrow \mathtt{Q}$ or IF P THEN Q, where P is a proposition that can contain a conjunction of n arbitrary attribute-value pairs, $\mathtt{P} = condition_1 \wedge \ldots \wedge condition_n$. It is important to observe that n is known as the rule length, and Q is the value of the categorical target attribute.

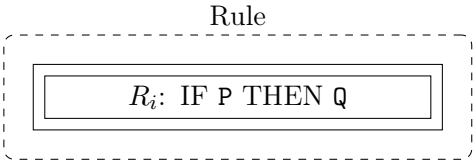

Fig. 1. Structure of an IF-THEN rule.

According to Mitchell [13], a possible approach to learn sets of rules involves learning a decision tree through an induction algorithm, such as ID3 [18], followed by a translation of such tree to an equivalent set of rules. A decision tree can be mapped to a set of rules, transforming each branch into a rule, i.e, each path from the root to one leaf corresponds to a rule. Fig. 2 presents a hypothetical example of decision tree.

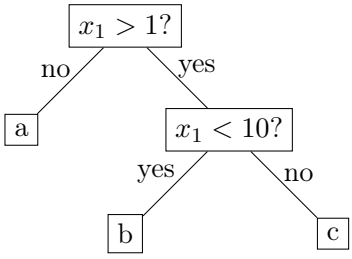

Fig. 2. Example of a decision tree. Each path from the root to a leaf can be written down as a conjunctive rule, composed of conditions defined by the decision nodes on the path.

The decision tree shown in Fig. 2 can be written down as the following decisions rule set:

- RULE 1: IF $x_1 > 1$ THEN $class = a$;
- RULE 2: IF $x_1 > 1 \wedge x_1 < 10$ THEN $class = b$;
- RULE 3: IF $x_1 > 1 \wedge x_1 >= 10$ THEN $class = c$;

As reported by [23], this approach produces rules that are unambiguous in the sense that the order they are executed does not matter. However, the rules are more complex than necessary. As we have just seen, we can obtain IF-THEN rules by learning a decision tree and converting it to rules. Another approach covers the direct rule learning. Decision rules learning works similarly to a decision tree except

that the rule induction does a depth-first search and generates one rule at a time, whilst the decision tree induction does a breadth-first search and generates all paths simultaneously [1].

Sequential covering algorithms use a popular technique for learning rule sets based on the strategy of learning one rule at a time, removing the data it covers, then iterating this process [23]. The following steps show a sketch of how a general set covering algorithm works [2]:

(i) Create a rule that covers some examples of a certain class and does not cover any examples of other classes,

(ii) remove covered examples from training data, and

(iii) if there are some examples not covered by any rule, go to step 1.

A prototypical sequential covering algorithm is described in Algorithm 1.

Algorithm 1 Sequential Covering algorithm

```
1: procedure SEQUENTIAL COVERING (target attributes, attributes, examples, threshold)
2:     learned rules ← {}
3:     rule ← LEARN-ONE-RULE()
4:     while PERFORMANCE(rules, examples) ≤ threshold do
5:         learnedRules ← learned rules + rule
6:         examples ← examples − {examples correctly}
7:     end while
8:     return learned rules
9: end procedure
```

As reported by Mitchell [13], LEARN-ONE-RULE must return a single rule that covers at least some of the *examples*. PERFORMANCE is a user-provided subroutine to evaluate the rule quality. This covering algorithm learns rules until it can no longer learn a rule whose performance is above the give *threshold*. Common evaluation functions include [13]:

- *Relative frequency.* Let n denote the number of examples the rule matches and let n_c denote the number of these that it classifies correctly. The relative frequency estimate of rule performance w is given by:

$$w = \frac{n_c}{n} \tag{1}$$

- *m-estimate of accuracy.* This accuracy estimate is biased toward the default accuracy expected for the rule. Let p be the prior probability that a randomly drawn example from the entire dataset will have the classification assigned by the rule. Let m be the weight, or equivalent number of examples for weighting this prior p. The m-estimate of rule accuracy is given by:

$$m_e = \frac{n_c + m_p}{n + m} \tag{2}$$

- *Entropy.* Entropy measures the uniformity of target function values for this set of examples. Let S be the set of examples that matches the rule preconditions. We

take the entropy negative so that better rules will have higher scores.

$$S = \sum_{i=1}^{n} p_i \log_2 p_i \tag{3}$$

where p is the probability that an event occurs and i is the index of the event.

2.2 Rule-driven adaptive devices

This subsection formally introduces the mathematical formalism proposed by José Neto [15]. Observe that the theory relies on an adaptive mechanism enclosing a non-adaptive rule-driven device, such that the latter may be enhanced in order to accommodate an adaptive behavior while preserving its integrity and original properties.

Definition 2.1 [rule-driven device] A rule-driven device is defined as $ND = (C, NR, S, c_0, A, NA)$, such that ND is a rule-driven device, C is the set of all possible configurations, $c_0 \in C$ is the initial configuration, S is the set of all possible input stimuli, $\epsilon \in S$, $A \subseteq C$ is the subset of all accepting configurations (respectively, $F = C - A$ is the subset of all rejecting configurations), NA is the set of all possible output stimuli of ND as a side effect of rule applications, $\epsilon \in NA$, and NR is the set of rules defining ND as a relation $NR \subseteq C \times S \times C \times NA$.

Definition 2.2 [rule] A rule $r \in NR$ is defined as $r = (c_i, s, c_j, z)$, $c_i, c_j \in C$, $s \in S$ and $z \in NA$, indicating that, as response to a stimulus s, r changes the current configuration c_i to c_j, processes s and generates z as output [15]. A rule $r = (c_i, s, c_j, z)$ is said to be compatible with the current configuration c if and only if $c_i = c$ and s is either empty or equals the current input stimulus; in this case, the application of a rule r moves the device to a configuration c_j, denoted by $c_i \Rightarrow^s c_j$, and adds z to the output stream.

Definition 2.3 [acceptance of an input stimuli stream by a rule-driven device] An input stimuli stream $w = w_1 w_2 \dots w_n$, $w_k \in S - \{\epsilon\}$, $k = 1, \dots, n$, $n \geq 0$, is accepted by a device ND when $c_0 \Rightarrow^{w_1} c_1 \Rightarrow^{w_2} \dots \Rightarrow^{w_n} c$ (in short, $c_0 \Rightarrow^w c$), and $c \in A$. Respectively, w is rejected by ND when $c \in F$. The language described by a rule-driven device ND is represented by $L(ND) = \{w \in S^* \mid c_0 \Rightarrow^w c, c \in A\}$.

Definition 2.4 [adaptive rule-driven device] A rule-driven device $AD = (ND_0, AM)$, such that ND_0 is a device and AM is an adaptive mechanism, is said to be adaptive when, for all operation steps $k \geq 0$ (k is the value of an internal counter T starting in zero and incremented by one each time a non-null adaptive action is executed), AD follows the behavior of an underlying device ND_k until the start of an operation step $k + 1$ triggered by a non-null adaptive action, modifying the current rule set; in short, the execution of a non-null adaptive action in an operation step $k \geq 0$ makes the adaptive device AD evolve from an underlying device ND_k to ND_{k+1}.

Definition 2.5 [operation of an adaptive device] An adaptive device AD starts its operation in configuration c_0, with the initial format defined as $AD_0 =$

$(C_0, AR_0, S, c_0, A, NA, BA, AA)$. In step k, an input stimulus move AD to the next configuration and starts the operation step $k+1$ if and only if a non-adaptive rule is executed; thus, being the device AD in step k, with $AD_k = (C_k, AR_k, S, c_k, A, NA, BA, AA)$, the execution of a non-null adaptive action leads to $AD_{k+1} = (C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA)$, in which $AD = (ND_0, AM)$ is an adaptive device with a starting underlying device ND_0 and an adaptive mechanism AM, ND_k is an underlying device of AD in an operation step k, NR_k is the set of non-adaptive rules of ND_k, C_k is the set of all possible configurations for ND in an operation step k, $c_k \in C_k$ is the starting configuration in an operation step k, S is the set of all possible input stimuli of AD, $A \subseteq C$ is the subset of accepting configurations (respectively, $F = C - A$ is the subset of rejecting configurations), BA and AA are sets of adaptive actions (both containing the null action, $\epsilon \in BA \cap AA$), NA, with $\epsilon \in NA$, is the set of all output stimuli of AD as side effect of rule applications, AR_k is the set of adaptive rules defined as a relation $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$, with AR_0 defining the starting behavior of AD, AR is the set of all possible adaptive rules for AD, NR is the set of all possible underlying non-adaptive rules of AD, and AM is an adaptive mechanism, $AM \subseteq BA \times NR \times AA$, to be applied in an operation step k for each rule in $NR_k \subseteq NR$.

Definition 2.6 [adaptive rules] Adaptive rules $ar \in AR_k$ are defined as $ar = (ba, c_i, s, c_j, z, aa)$ indicating that, as response to an input stimulus $s \in S$, ar initially executes the prior adaptive action $ba \in BA$; the execution of ba is canceled if this action removes ar from AR_k; otherwise, the underlying non-adaptive rule $nr = (c_i, s, c_j, z)$, $nr \in NR_k$ is applied and, finally, the post adaptive action $aa \in AA$ is applied [15].

Definition 2.7 [adaptive function] Adaptive actions may be defined as abstractions named adaptive functions, similar to function calls in programming languages [15]. The specification of an adaptive function must include the following elements: *(a)* a symbolic name, *(b)* formal parameters which will refer to values supplied as arguments, *(c)* variables which will hold values of applications of elementary adaptive actions of inspection, *(d)* generators that refer to new value references on each usage, and *(e)* the body of the function itself.

Definition 2.8 [elementary adaptive actions] Three types of elementary actions are defined in order to perform tests on the rule set or modify existing rules, namely:

(i) inspection: the elementary action does not modify the current rule set, but allows inspection on such set and querying rules that match a certain pattern. It employs the form $?\langle\text{pattern}\rangle$.

(ii) removal: the elementary action removes rules that match a certain pattern from the current rule set. It employs the form $-\langle\text{pattern}\rangle$. If no rule matches the pattern, nothing is done.

(iii) insertion: the elementary action adds a rule that match a certain pattern to the rule set. It employs the form $+\langle\text{pattern}\rangle$. If the rule already exists in the rule set, nothing is done.

Such elementary adaptive actions may be used in the body of an adaptive function, including rule patterns that use formal parameters, variables and generators available in the function scope.

Fig. 3 presents the general concept of a set of rules enhanced with adaptive actions being mapped to adaptive functions. Note that B and A denote adaptive actions to be triggered before and after the rule application, respectively.

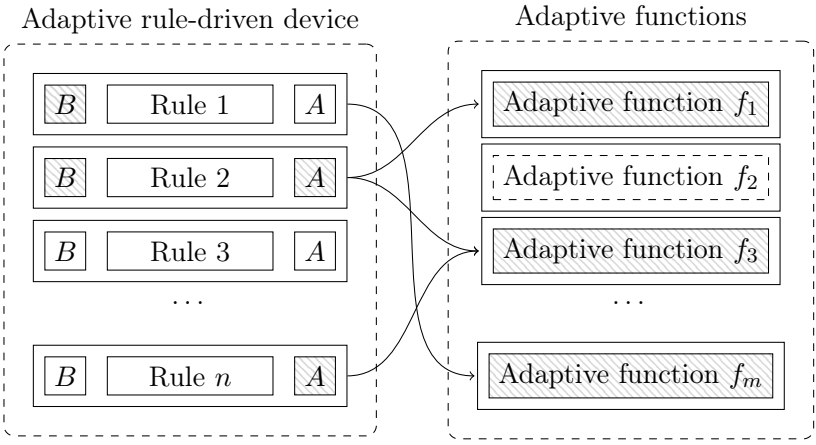

Fig. 3. Set of rules enhanced with adaptive actions being mapped to adaptive functions.

3 An adaptivity-based rule learning technique

Our approach for an adaptivity model is inspired on the sequential covering description presented in Algorithm 1, with subtle yet significant differences. According to Algorithm 1, subroutine LEARN-ONE-RULE accepts a set of positive and negative training examples as input and returns as output a single rule that covers many of the positive examples and few of the negative ones. We propose a modification to this subroutine, such that it now returns different rules that cover both positive and negative examples (we have also renamed such subroutine to LEARN-RULES). We also treat the set of examples differently, such that LEARN-RULES is invoked on all available training examples. Additionally, it removes positive or negative examples completely covered by the rule it learns, according to some heuristic (i.e, whenever *relative frequency* is equal to 1).

The sequential covering strategy uses a greedy approach that takes the best local actions where it iteratively learns a single rule, so that each learned rule is added to the rule base. Generally, those rules being added in each iteration keep being part of the rule base until the end of the learning process. Thus, our proposal aims at modifying the sequential covering strategy, such that the new technique is able to review already learned rules through adaptive functions. In each step, it is possible to decide among three options:

(i) add a new rule that improves the prediction capability of the rule base,

(ii) add the best rule that replaces one or more rules of the rule base, as an attempt to increase the prediction capability, or

(iii) the rule base it is not updated.

Algorithm 2 learns rules until all instances have been covered by the rule set, or there are no more attributes to be added.

Algorithm 2 Adaptive algorithm

```
1: procedure MAIN(attributes, D)
2:     learned rules ← {}
3:     examples ← D
4:     attributes ← INFORMATION-GAIN(attributes, D)
5:     while PERFORMANCE(attributes, examples) ≤ threshold do
6:         candidate rules ← {}
7:         learnedRules ← LEARN-RULES()
8:         examples ← examples − {correctly classified examples}
9:     end while
10:    return learned rules
11: end procedure
```

The main loop searches for a default rule R_i in a set of rules AR_k. Default rule R_i is defined as $R_i = \texttt{P} \rightarrow \texttt{Q (w)}[A_i], i = 1 \ldots n$, where $[A_i]$, is an adaptive function and it is defined in Algorithm 3.

Algorithm 3 Adaptive Function A_i

```
1: procedure A_i(r_j, P, Q, fr)
2:     −[P → Q(w)[A_i]]:
3:     +[r_j(fr)[A_i]]:
4: end procedure
```

A possible approach to implement the LEARN-RULES subroutine - see Algorithm 4, consists of organizing the hypothesis space search similarly to ID3 algorithm's behavior [18], but restricting the search to the most promising tree branch at each step. Our approach uses a breadth-first search to construct the next rule. Since the rule consequent must be the given class (in this case, positive or negative), only the antecedent needs to be constructed; this is achieved by starting with an empty antecedent and iteratively adding an *attribute-value* pair for all attribute values.

Algorithm 4 Procedure to learn rules

```
1: procedure LEARN-RULES(examples)
2:     canditate rules ← GET-CANDIDATES()
3:     while i = 1 ≤| CANDIDATE RULES | do
4:         rule ← EVALUATE(r_j)
5:
6:         if rule ≠ null then
7:             In this case, the performance of the rule is compared:
                   (fr > w) = { 0 then do not anything
                              { 1 then apply the rule
8:         else
9:             A new rule [R_i : P → Q(fr)[A_n]] is added in AR_k.
10:        end if
11:    end while
12:    return learned rules
13: end procedure
```

An evaluation function is used to rate candidate rules, called EVALUATE(r_j). Each candidate rule r_j is defined as $r_j = \texttt{P} \rightarrow \texttt{Q (fr)}$, where fr is the calculated

frequency. This breadth-first search continues until the resulting rule is specific enough. The resulting classifier can contain three types of rules: *(a)* rules with 100% accuracy (higher rank), *(b)* rules with good accuracy (lower rank), i.e. $< 100\%$ and $> 50\%$, and *(c)* rules with poor accuracy (mean rank), i.e. $<= 50\%$. When searching for an applicable rule, the algorithm goes over the rules in a topdown fashion starting with the primary rules (higher rank) until reaching rules with a poor rank. Whenever two or more rules have identical performance then the algorithm favors rules with the least number of terms in their `P`.

4 An illustrative example

This section presents a didactic example in order to illustrate the rule learning process through our adaptive algorithm. We have selected a well-known dataset proposed by [18], named *Weather dataset* [5] , described in Table 1. In general, the approach assumes that:

(i) the algorithm obtains a dataset containing training examples as input (e.g, the ones described in Table 1),

(ii) the dataset consists of nominal attributes (e.g, the ones described in Table 2),

(iii) the attributes hold a predefined order (e.g, the ones described in Table 3), and

(iv) each instance is mapped to exactly one element from the set of positive and negative class labels.

Given requirement #4, we begin by considering classification problems using only two classes of interest.

In this case, the learner starts with a simple rule representation in which each `P` consists of a conjunction of constraints on the instance attributes. In particular, let each condition `P` be a vector of four constraints, referring to the values of attributes *outlook*, *temperature*, *humidity* and *wind*. For each attribute, the rule will either:

(i) indicate with the wildcard symbol ? that any value is acceptable for this attribute, or

(ii) specify a single required value (e.g., *sunny*) for the attribute.

4.1 First step through the loop

The frequency is calculated for each attribute value (*outlook*). Then the relative frequency is computed for all candidate rules in the form $\texttt{P} \rightarrow \texttt{Q (w)}$.

According to Table 4, we can get a frequency table for *outlook* values. Based on the frequency tables, the relative frequency for each candidate rule can be calculated, as seen in Table 5. Each candidate rule is evaluated according to the current rule set in AR_k. The achievement of this step is shown in Table 6.

This completes the first pass through the inner loop in the sequential covering algorithm. Since there is still one uncovered remaining instance, another step is

[5] http://storm.cis.fordham.edu/ gweiss/data-mining/weka-data/weather.arff

id	Outlook	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	False	No
2	Sunny	Hot	High	True	No
3	Overcast	Hot	High	False	Yes
4	Rain	Mild	High	False	Yes
5	Rain	Cool	Normal	False	Yes
6	Rain	Cool	Normal	True	No
7	Overcast	Cool	Normal	True	Yes
8	Sunny	Mild	High	False	No
9	Sunny	Cool	Normal	False	Yes
10	Rain	Mild	Normal	False	Yes
11	Sunny	Mild	Normal	True	Yes
12	Overcast	Mild	High	True	Yes
13	Overcast	Hot	Normal	False	Yes
14	Rain	Mild	High	True	No

Table 1
Weather dataset.

Outlook	Temperature	Humidity	Wind	Play?
Sunny	Hot	High	False	No
Overcast	Mild	Normal	True	Yes
Rain	Cool			

Table 2
Dataset attributes and their corresponding values.

performed in order to generate additional rules. After the first step the rule set AR_5 is as seen in Table 7.

When the five candidate rules are chosen, rule R_5 has the maximum relative frequency, i.e. $w = 1$. Since the maximum value is reached, the learning process for this rule is complete. So there is no need for adaptive functions to improve the rule performance. Also, all four instances with the *attribute-value* pair *outlook* = *overcast* are deleted from the training set.

Ranking	Attribute	Gain
1	Outlook	0.2467
2	Humidity	0.1518
3	Windy	0.0481
4	Temperature	0.0292

Table 3
Ordered attributes using information gain.

Sunny	Overcast	Rain	Play?
2	4	3	Yes
3	0	2	No
5	4	5	Total

Table 4
Frequency for attribute *outlook*.

Input	Relative frequency
$r_1 : (Sunny, ?, ?, ?) \rightarrow$ Yes	$fr = 0.4$
$r_2 : (Sunny, ?, ?, ?) \rightarrow$ No	$fr = 0.6$
$r_3 : (Rain, ?, ?, ?) \rightarrow$ Yes	$fr = 0.6$
$r_4 : (Rain, ?, ?, ?) \rightarrow$ No	$fr = 0.4$
$r_5 : (Overcast, ?, ?, ?) \rightarrow$ Yes	$fr = 1.0$

Table 5
Candidate rules for attribute *outlook*.

4.2 Second step through the loop

The frequency is calculated for each attribute value (*humidity*). Then the relative frequency is computed for all candidate rules.

According to Table 8, we can obtain a frequency table for *humidity* values. Based on the frequency tables, the relative frequency for each candidate rule can be calculated, as seen in Table 9. Each candidate rule is evaluated according to the current rule set AR_k. The brief of such step is presented in Table 10.

After the second step, the rule set AR_{10} is as described in Table 11.

Similarly to the previous step, rules R_1 and R_2 have the maximum relative frequency, so the learning process for these rules is complete. All instances not covered by them are deleted from the training set. Observe that R_4 and R_6 are

r_j	EVALUATE(r_j)	Action	Update AR_k
r_1	$rule \leftarrow null$	add $R_1 \leftarrow r_1$	$AR_0 \rightarrow AR_1$
r_2	$rule \leftarrow null$	add $R_2 \leftarrow r_2$	$AR_1 \rightarrow AR_2$
r_3	$rule \leftarrow null$	add $R_3 \leftarrow r_3$	$AR_2 \rightarrow AR_3$
r_4	$rule \leftarrow null$	add $R_4 \leftarrow r_4$	$AR_3 \rightarrow AR_4$
r_5	$rule \leftarrow null$	add $R_5 \leftarrow r_5$	$AR_4 \rightarrow AR_5$

Table 6
Constructing our rule set, step 1.

ID	Rule	w	Adaptive Function
R_1	$(Sunny, ?, ?, ?) \rightarrow$ Yes	(0.4)	$[A_1]$
R_2	$(Sunny, ?, ?, ?) \rightarrow$ No	(0.6)	$[A_2]$
R_3	$(Rain, ?, ?, ?) \rightarrow$ Yes	(0.6)	$[A_3]$
R_4	$(Rain, ?, ?, ?) \rightarrow$ No	(0.4)	$[A_4]$
R_5	$(Overcast, ?, ?, ?) \rightarrow$ Yes	(1.0)	*null*

Table 7
Rule set AR_5 after step 1.

Sunny/High	Sunny/Normal	Rain/High	Rain/Normal	Play?
0	2	1	2	Yes
3	0	1	1	No
3	2	2	3	Total

Table 8
Frequency for attributes outlook/humidity.

currently conflicting rules.

4.3 Third step through the loop.

The frequency is calculated for each attribute value (*wind*). Then the relative frequency is computed for all candidate rules.

In accordance with Table 12, we can obtain a frequency table for *humidity* values. Based on the frequency tables, the relative frequency for each candidate rule can be calculated, as seen in Table 13. Each candidate rule is evaluated according to the current rule set AR_k. This step is displayed in Table 14.

The resulting rule set AR_{13} is presented in Table 15. As all rules reached $w = 1.0$,

Input	Relative frequency
$r_1 : (Sunny, ?, High, ?) \rightarrow$ No	$fr = 1.0$
$r_2 : (Sunny, ?, Normal, ?) \rightarrow$ Yes	$fr = 1.0$
$r_3 : (Rain, ?, High, ?) \rightarrow$ Yes	$fr = 0.7$
$r_4 : (Rain, ?, High, ?) \rightarrow$ No	$fr = 0.3$
$r_5 : (Rain, ?, Normal, ?) \rightarrow$ No	$fr = 0.5$
$r_6 : (Rain, ?, Normal, ?) \rightarrow$ Yes	$fr = 0.5$

Table 9
Candidate rules for attribute humidity.

r_j	EVALUATE(r_j)	Action	Update AR_k
r_1	$rule \leftarrow R_2$	apply $[A_2]$ and $R_2 \leftarrow r_1$	$AR_5 \rightarrow AR_6$
r_2	$rule \leftarrow R_1$	apply $[A_1]$ and $R_1 \leftarrow r_2$	$AR_6 \rightarrow AR_7$
r_3	$rule \leftarrow R_3$	apply $[A_3]$ and $R_3 \leftarrow r_3$	$AR_7 \rightarrow AR_8$
r_4	$rule \leftarrow R_4$	$fr \leq w$, do not anything	–
r_5	$rule \leftarrow R_4$	apply $R_4 \leftarrow r_5$	$AR_8 \rightarrow AR_9$
r_6	$rule \leftarrow null$	add $R_6 \leftarrow r_5$	$AR_9 \rightarrow AR_{10}$

Table 10
Constructing our rule set, step 2.

ID	Rule	w	Adaptive Function
R_1	$(Sunny, ?, Normal, ?) \rightarrow$ Yes	$(1, 0)$	$null$
R_2	$(Sunny, ?, High, ?) \rightarrow$ No	$(1, 0)$	$null$
R_3	$(Rain, ?, High, ?) \rightarrow$ Yes	$(0, 7)$	$[A_3]$
R_4	$(Rain, ?, Normal, ?) \rightarrow$ No	$(0, 5)$	$[A_4]$
R_5	$(Overcast, ?, ?, ?) \rightarrow$ Yes	(1.0)	$null$
R_6	$(Rain, ?, Normal, ?) \rightarrow$ yes	$(0, 5)$	$[A_6]$

Table 11
Rule set AR_{10} after step 2.

Rain/High /False	Rain/High /True	Rain/Normal /False	Rain/Normal /True	Play?
0	1	1	0	Yes
1	0	0	1	No
1	1	1	1	Total

Table 12
Frequency for attributes outlook/humidity/wind.

Input	Relative frequency
$r_1 : (Rain, ?, High, False) \rightarrow$ Yes	$fr = 1.0$
$r_2 : (Rain, ?, High, True) \rightarrow$ No	$fr = 1.0$
$r_3 : (Rain, ?, Normal, False) \rightarrow$ Yes	$fr = 1.0$
$r_4 : (Rain, ?, Normal, True) \rightarrow$ No	$fr = 1.0$

Table 13
Candidate rules for attribute wind.

r_j	EVALUATE(r_j)	Action	Update AR_k
r_1	$rule \leftarrow R_3$	apply $[A_3]$ and $R_3 \leftarrow r_1$	$AR_{10} \rightarrow AR_{11}$
r_2	$rule \leftarrow null$	add $R_7 \leftarrow r_2$	$AR_{11} \rightarrow AR_{12}$
r_3	$rule \leftarrow R_6$	apply $[A_6]$ and $R_6 \leftarrow r_3$	$AR_{12} \rightarrow AR_{13}$
r_4	$rule \leftarrow R_4$	apply $[A_4]$ and $R_4 \leftarrow r_4$	$AR_{13} \rightarrow AR_{14}$

Table 14
Constructing our rule set, step 3.

that means that all samples from the training set (Table 1) are covered, and thus there is no need for another iteration.

The algorithm mainly focus on maximizing the rule accuracy, even when the discovered rules covers one sample from the training data. Further studies are needed regarding potential data overfit on large rule sets.

5 Final remarks

This paper presented an approach towards the definition of an adaptive learning algorithm for rule inference. Our approach is based on modifications of conventional sequential covering strategies in order to adapt acquired knowledge during the learning process. Studies indicate that adaptive techniques confer a more adequate rule

ID	Rule	w	Adaptive Function
R_1	$(Sunny, ?, Normal, ?) \rightarrow Yes$	$(1, 0)$	*null*
R_2	$(Sunny, ?, High, ?) \rightarrow No$	$(1, 0)$	*null*
R_3	$(Rain, ?, High, False) \rightarrow Yes$	(1.0)	*null*
R_4	$(Rain, ?, High, True) \rightarrow No$	(1.0)	*null*
R_5	$(Overcast, ?, ?, ?) \rightarrow Yes$	(1.0)	*null*
R_6	$(Rain, ?, Normal, False) \rightarrow Yes$	(1.0)	*null*
R_7	$(Rain, ?, Normal, True) \rightarrow No$	(1.0)	*null*

Table 15
Set Rules AR_{13}

set fitting. Additional advantages are noteworthy:

- *simplicity of rule generation*, in which only a single performance measure is computed in order to decide the rule significance,
- *comprehensible rule set for decision making*, specially for domains that require immediate interpretation, such as medical applications, and
- *compact and efficient model representation*, such that the rule set growth or reduction is dictated by calls to adaptive functions during construction time.

As a means to improve our proposal, as well as the current adaptive algorithm, we compiled a list of features and enhancements, as follows, to be contemplated in a near future:

- *Search space reduction for candidate rules.* When considering large dimensional datasets, the numbers of candidate rules might significantly increase, and thus impose operational limits to certain applications. A potentially heuristic-based mechanism to reduce the search space might allow better handling of huge datasets.
- *Numerical attribute handling and support for noisy datasets.* A dataset is said to be noisy when it contains incomplete attributes and missing values. An approach to handle and potentially predict completeness of partial information might provide better model convergence. Additionally, the algorithm must offer a discretization strategy for covering attributes in a continuous domain.
- *Resolution policy for conflicting rules.* The algorithm must provide a policy for resolving conflicts in the rule set being constructed, e.g, by adopting class labels with the largest linked frequency as decision criterion.
- *Tiebreaker mechanism.* The algorithm must provide a mechanism to decide whether a rule should be replaced by another, when both have the same frequencies.

- *Pruning support.* As a means to avoid a rapid growth of nonessential candidate rules (leading to an inevitable combinatorial explosion), the algorithm must support pruning methods in order to keep the rule set cardinality at an acceptable number.

We are working on comparative experiments with other sequential coverage algorithms, in order to verify the model accuracy. Prospectively, we aim at studying the algorithm adaptability when new examples are available, as well as certain cases in which missing values must be considered. However, preliminary results indicate a significant improvement in accuracy, as well as a more comprehensible rule set. Adaptivity poses as an interesting phenomenon to be explored in rule learning, as the rule set construction process potentially accommodates new contexts over time.

References

[1] Alpaydin, E., "Introduction to Machine Learning," Adaptive Computation and Machine Learning, MIT Press, 2014.

[2] Berka, P. and J. Rauch, *Machine learning and association rules*, in: *Proceedings of the 19th International Conference on Computational Statistics*, 2010.

[3] Bishop, C. M., "Pattern recognition and machine learning," Springer, 2006, 1 edition.

[4] Cendrowska, J., *PRISM: An algorithm for inducing modular rules*, International Journal of Man-Machine Studies **27** (1987), pp. 349–370.

[5] Clark, P. and T. Niblett, *The cn2 induction algorithm*, Machine Learning **3** (1989), pp. 261–283.

[6] Costa, E. R., A. R. Hirakawa and J. J. Neto, *An adaptive alternative for syntactic pattern recognition*, in: *Proceeding of 3rd International Symposium on Robotics and Automation, ISRA 2002*, 2002, pp. 409–413.

[7] Doy, B. T. M., D. F. Souza and R. G. Jankauskas, *Aocr: adaptive optical character recognition*, in: *Quarto Workshop de Tecnologia Adaptativa – WTA 2010*, 2010, pp. 50–54.

[8] Elgibreen, H. A. and M. S. Aksoy, *Rules-TL: a simple and improved rules algorithm for incomplete and large data*, Journal of Theoretical and Applied Information Technology **47** (2013), pp. 28–40.

[9] Fürnkranz, J., *Separate-and-conquer rule learning*, Artificial Intelligence Review **13** (1999), pp. 3–54.

[10] Ganzeli, H. S., J. G. Bottesini, L. O. Paz and M. F. S. Ribeiro, *Skan: Skin scanner, software para o reconhecimento de câncer de pele utilizando técnicas adaptativas*, in: *Quarto Workshop de Tecnologia Adaptativa – WTA 2010*, 2010, pp. 22–28.

[11] Hirakawa, A. R., A. M. Saraiva and C. E. Cugnasca, *Autômatos adaptativos aplicados em automação e robótica*, IEEE Latin America **5** (2007), pp. 539–543.

[12] Michalski, R. S., *On the quasi-minimal solution of the general covering problem*, in: *Proceedings of the 5th International Symposium on Information Processing*, Bled, Yugoslavia, 1969, pp. 125—-128.

[13] Mitchell, T., "Machine learning," McGraw-Hill Science, 1997, 1 edition.

[14] Neto, J. J., *Solving complex problems with adaptive automata*, in: S. Yu and A. Paun, editors, *Implementation and Application of Automata 5th International Conference*, Lecture Notes in Computer Science **2088**, 2000.

[15] Neto, J. J., *Adaptive rule-driven devices: General formulation and case study*, in: B. Watson and D. Wood, editors, *Implementation and Application of Automata 6th International Conference*, Lecture Notes in Computer Science **2494**, 2001, pp. 234–250.

[16] Pistori, H. and J. J. Neto, *Adaptree: Proposta de um algoritmo para indução de Árvores de decisão baseado em técnicas adaptativas*, in: *Anais da Conferência Latino Americana de Informática – CLEI 2002*, 2002.

[17] Pistori, H. and J. J. Neto, *An experiment on handshape sign recognition using adaptive technology: Preliminary results*, in: *XVII Brazilian Symposium on Artificial Intelligence – SBIA 04*, 2004.

[18] Quinlan, J. R., *Induction of decision trees*, Machine Learning **1** (1986), pp. 81–106.

[19] Stahl, F. and M. Bramer, *Random prism: An alternative to random forests*, in: *Research and Development in Intelligent Systems XXVIII*, Springer London, 2011 pp. 5–18.

[20] Stange, R. L. and J. J. Neto, *Applying adaptive technology in machine learning*, IEEE Latin America **12** (2014), pp. 1298–1306.

[21] Stange, R. L. and J. J. Neto, *Learning decision rules using adaptive technologies: a hybrid approach based on sequential covering*, Procedia Computer Science **109** (2017), pp. 1188–1193.

[22] Thabtah, F., I. Qabajeh and F. Chiclana, *Constrained dynamic rule induction learning*, Expert Systems with Applications **63** (2016), pp. 74–85.

[23] Witten, I. H., E. Frank and M. A. Hall, "Data Mining: Practical Machine Learning Tools and Techniques," Morgan Kaufmann, 2011.