SILEA: a System for Inductive LEArning

Ahmet Aksoy
Computer Science and Engineering Department
University of Nevada, Reno
Email: aksoy@nevada.unr.edu

Mehmet Hadi Gunes
Computer Science and Engineering Department
University of Nevada, Reno
Email: mgunes@unr.edu

***Abstract*—This paper presents SILEA (a System for Inductive LEArning), an efficient inductive learning algorithm for rule extraction. SILEA is a covering algorithm which extracts IF-THEN rules from a collection of examples in a reliable way. The algorithm eliminates exhaustive feature selection by reducing the number of attributes(features) to be considered for each necessary iteration of rule extraction. For every iteration, depending on the number of conditions, it prioritizes numerous attributes over the others to reduce the large number of attribute combinations. This prioritization, however, needs to be done attentively to prevent loss in performance or possibly improve it. SILEA employs the entropy measure for such purpose. As the entropy value decreases for an attribute, its predictability increases. SILEA favors the lower entropy-valued attributes for rule extraction. Another important factor in preserving or improving the performance of the algorithm is the rule extraction and selection procedure. SILEA induces every possible rule for the given combination and selects the most classifying ones among them. It also eliminates rules which might become obsolete by the existence of rules with higher classification performance. In conjunction of these two features, i.e., entropy based attribute prioritization and redundant rule elimination, SILEA extracts rules both accurately and efficiently. The paper describes how the algorithm functions along with its features and discusses its performance compared to some of the well-known algorithms in the field on a number of different data sets.**

I. Introduction

Machine learning techniques have been very helpful in terms of knowledge extraction from examples in an automatic way [18]. One of the most preferred machine learning techniques undoubtedly is inductive learning [16]. Inductive learning is a way of reaching general rules from specific examples [17]. The reasons for inductive learning algorithms to be preferred are their simplicity, speed and accuracy [16]. The categories of inductive learning can be listed as divide-and-conquer methods and covering methods [24]. Decision tree-based approaches of divide-and-conquer methods are efficient but not always reliable in terms of the generality of the rules they generate. Covering-method algorithms provide more flexibility and generality but they have higher complexity. The proposed SILEA (a System for Inductive LEArning) aims to provide both efficiency and generality in the rules it generates.

In this paper, we present a new inductive learning algorithm called SILEA which generates a set of IF-THEN rules in an efficient way. It employs a feature selection technique similar to that of Sequential Forward Selection [27] to decrease the number of attribute-value pairs that are to be considered. An attribute-value pair defines the affiliation of an attribute and the specific value(s) it can take [2]. Sequential Forward Selection prioritizes certain attributes over the others by an objective function [3]. Similarly, SILEA makes sure that these biases are made on attributes with higher metric values than the other attributes. The metric used for this purpose is the entropy measure. This approach allows SILEA to avoid consideration of enormous amount of possible combinations for each iteration. In addition, the extraction process that SILEA employs eliminates unnecessary comparisons needed for rule extraction. It extracts all possible rules for each considered combination and selects the most classifying ones among them. It excludes the rules which might become obsolete due to the existence of more classifying rules. This approach assures the extraction of the most general rules for the considered combination of attributes.

In order to asses SILEA's complexity and performance, we compared it with some of the well-known algorithms in the field. SILEA was able to considerably reduce the number of attribute combinations, i.e., from O(n^3) to O(n^2) in the worst case, and have a smaller run-time complexity. The algorithm was also able to perform better than the others on averages of all the analyzed datasets.

The rest of the paper is organized as follows: Section II describes the related work. Section III explains the proposed SILEA algorithm. Section IV presents an illustrative problem to explain how the algorithm works. Section V shows performance results of the algorithm compared to similar algorithms. Section VI concludes the paper.

II. Related Work

There exist many divide-and-conquer and covering type inductive learning algorithms which induce knowledge from examples. Inductive learning forms a knowledge base from a given set of examples where each example contains a number of attribute values along with a class [1].

ID3 is a divide-and-conquer algorithm introduced by Quinlan in 1987 which uses training examples to generate a decision tree [23]. It selects nodes according to their entropies in order to construct a tree more efficiently [22]. ID3, however, lacks in terms of the generality of the trees it generates [5]. Many covering-method algorithms have used this vulnerability to their advantages by introducing algorithms which focus more on the generality of the rules they induce.

C4.5 is another well known divide-and-conquer algorithm proposed by Quinlan in 1993 [25]. This improved algorithm,

unlike ID3, can handle continuous attributes. Another improvement was pruning. ID3 is more sensitive to noise in data and in order to prevent the tree from over-fitting the data, C4.5 prunes the tree by eliminating the sections of the tree which contribute little to the classification of data [12].

AQ is a covering algorithm introduced in 1969 by Michalski [14]. AQ was initially introduced to solve the boolean function satisfiability problem. However, it later has been adapted to solve the covering problem. The algorithm has been applied to several problems such as the generation of individuals within an evolutionary computation network. The algorithm has been improved several times but has not been used widely mainly due to its high complexity [20].

CN2 is a covering algorithm introduced in 1989 by Peter Clark and Tim Niblett [6]. CN2 algorithm takes advantage of ID3's noisy data handling approach along with the flexibility of AQ family algorithms. In CN2, rule forming procedure is terminated by the use of a heuristic function based on an estimate of the noise observed in the data. As a result, CN2 might generate rules which do not necessarily classify all the training examples but expected to perform well on new data.

RULES3 is also a covering algorithm introduced in 1995 by Pham and Aksoy. RULES3 is an automatic rule extraction system which was released as an advancement to its predecessors, RULES1 and RULES2. RULES3, in addition to its predecessors, introduces two new features: 1) it allows the user to set precision of rules and 2) it provides more generality to the rules it generates [17].

RULES3-Plus was released in 1997 by Pham and Dimov [22] to overcome the issue of exhaustive searching procedure that RULES3 employs [17]. RULES3-Plus provides two advantages over its predecessor, 1) it adopts a more efficient rule searching procedure and 2) it uses the h-measure metric for selecting attributes [17], [22].

Several algorithms, based on RULES3 and RULES3-Plus, have been implemented. These algorithms have introduced many improvements to the base algorithms. For instance, RULES4 algorithm has the ability to extract rules incrementally [20]. RULES5 employs a new method to handle continuous attributes and to extract rules [18]. RULES6 also introduces a new method for continuous attribute handling along with a noise-tolerant rule extraction technique [16]. RULES-F algorithm, in addition to handling continuous attributes, can generate accurate and compact fuzzy models which allows it to handle continuous classes as well [19]. RULES3-EXT is able to perform attribute re-ordering and fire rules partially if the extracted rules are not able to classify new examples [13]. RULES-TL uses transfer learning by collecting knowledge from agents in different domains which helps reduce the search time [7]. RULES-IT algorithm is the incremental version of RULES-TL algorithm which also transfers rules from different domains to improve its performance [8].

In this paper, we propose improvements compared to the RULES3, the base algorithm over which the later approaches built on. Hence, such improvements can be incorporated into our SILEA algorithm as well.

III. Proposed Algorithm

SILEA is an inductive learning algorithm which aims to generate rules from a dataset in both an efficient and an accurate manner. In order to achieve these aims, the algorithm employs two important characteristics, namely, feature selection and rule extraction.

The feature selection approach that SILEA employs is similar to the Sequential Forward Selection method. To avoid consideration of every possible combination of attributes, the algorithm in each iteration fixes $n_c - 1$ number of attributes and takes the combinations of these pre-selected attributes with the remaining ones. However, to avoid performance drop, it needs to select these attributes attentively. Otherwise, it is possible for the algorithm to miss more optimal attribute combinations which would help generate more general rules. This is why SILEA uses the entropy measure to decide which attributes to select in each iteration of the rule extraction. It favors those with lower entropy values to assure the selection of more information-gaining attributes. This helps SILEA to generate as general rules as possible for a given iteration.

Second approach is regarding the rule extraction system of the algorithm. After deciding on which attributes to consider for rule extraction, SILEA extracts all possible rules from a dataset. For each iteration, SILEA is able to generate all of the rules for the given number of conditions with a single visit of every example in the dataset. Each potential rule that SILEA forms from the selected attributes are considered to be rules unless it contradicts with other examples. Contradiction occurs when the formed attribute combination value(s) belong to more than one class among the examples in the dataset. After the extraction, the rule selection phase of SILEA eliminates obsolete rules. Obsolete rules are the ones that can be replaced by other rules with higher occurrences. After selecting the rules, it discards the examples that can be classified by these rules. SILEA stops rule extraction when it is able to classify all of the examples in the dataset with the selected rules.

SILEA induces rules from a set of examples within a dataset as presented in Algorithm 1 through Algorithm 4. Each example in a dataset contains a number of attributes and a class [1]. A single or a combination of attributes is considered a condition. The number of attributes within a condition could vary between one and n_a (total number of attributes in an example). After data initialization in Section III-A, feature selection and rule extraction procedures of SILEA are explained in Section III-B and Section III-C.

A. Data Initialization

To ease the selection process prior to feature selection and rule extraction, the ***SortFeatures*** function is executed to sort attributes according to their entropy values from the lowest to the highest (Alg1-Ln1). SILEA quantizes numerical attributes by defining and setting ranges for their values (Alg1-Ln2). For the quantization process, the algorithm executes ***QuantizeAttributes*** function which finds the ranges of each numerical attribute (Alg2-Ln1) by determining the minimum and maximum values of the attribute (Alg2-Ln2-3) and dividing

Algorithm 1: ***SILEA*** rule forming procedure

1 ***SortFeatures***(*Examples*);
2 ***QuantizeAttributes***(*Examples*, *NoOfRanges*);
3 $SelectedRules = \emptyset$;
4 $n_c = 0$; // default
5 $UnclassifiedExamples = Examples$;
6 **while** *UnclassifiedExamples !=* $\emptyset$ **do**
7 $n_c = n_c + 1$;
8 $Blacklist = \emptyset$;
9 $PotentialList = \emptyset$;
10 **for** *each Example* $\in$ *Examples* **do**
11 *FormedRules* = ***GenerateFormedRules***(*Example*, n_c);
12 **for** *each Rule* $\in$ *FormedRules* **do**
13 **if** *Rule* $\notin$ *Blacklist* **then**
14 **if** *Rule* $\notin$ *PotentialList* **then**
15 $PotentialList = PotentialList \cup Rule$;
16 $Rule.occurrence = 1$;
17 **else if** *Rule* $\in$ *PotentialList* **and** *Rule.class* = *Example.class* **then**
18 $Rule.occurrence = Rule.occurrence + 1$;
19 **else if** *Rule* $\in$ *PotentialList* **and** *Rule.class !=* *Example.class* **then**
20 $PotentialList = PotentialList - Rule$;
21 $Blacklist = Blacklist \cup Rule$;

Algorithm 2: ***QuantizeAttributes***(*Examples*, *NoOfRanges)*

1 **for** *each* $Attribute_i \in Examples$ **do**
2 Min = ***FindMin***$(Attribute_i)$;
3 Max = ***FindMax***$(Attribute_i)$;
4 $Range = (Max - Min)/NoOfRanges$;
5 **for** *each* $Example \in Examples$ **do**
6 **if** $Example.Attribute_i <= Min$ **then**
7 $Example.Attribute_i = 1$;
8 **else if** $Example.Attribute_i > Max$ **then**
9 $Example.Attribute_i = NoOfRanges$;
10 **else**
11 $Example.Attribute_i = \lceil (Example.Attribute_i)/Range \rceil$;

Algorithm 3: ***GenerateFormedRules***(*Example*, n_c)

1 $FormedRules = FixedAttributes = \emptyset$;
2 **for** $i = 1$, $i{+}{+}$, ***while*** $i <= n_c - 1$ **do**
3 $FixedAttributes = FixedAttributes \cup Example.attribute_i$;
4 **for** $i = 1$, $i{+}{+}$, ***while*** $i <= n_a - (n_c - 1)$ **do**
5 $FormedRule = FixedAttributes \cup Example.attribute_i \cup Example.class$;
6 $FormedRules = FormedRules \cup FormedRule$;
7 **return** $FormedRules$;

their difference by the number of quantization levels provided by the user (Alg2-Ln4). Then, for each example (Alg2-Ln5), the range that the corresponding attribute value belongs to is calculated and assigned (Alg2-Ln6-12). Note that, we quantize values outside the range as min or max. After the quantization process, the algorithm goes into a loop which is executed until there are no more unclassified examples left in the dataset (Alg1-Ln6).

B. Feature Selection Procedure

SILEA tries to extract rules from an example set according to the unique correspondence of attribute-value pairs and their associated classes. It starts with the minimum number of conditions set by the user, and as needed, it keeps incrementing them until it reaches the maximum, which is equivalent to the number of attributes n_a.

An important feature of SILEA for optimizing the performance and accuracy of the classification while increasing efficiency is to prioritize certain attributes over the others. It employs a feature selection technique similar to that of Sequential Forward Selection [27] to immensely decrease the number of combinations to be dealt with. The criteria for such selection is decided based on the entropy of the attributes [26].

SILEA selects the attributes according to their entropies from the smallest to the highest, in other words, from the most information-gaining to the least. The entropy of i^{th} attribute A_i, i.e., $E(A_i)$ is

$$\sum_{j=1}^{|S_i|} \frac{|S_{ij} \in A_i|}{|I|} \left[-\sum_{k=1}^{|C|} \frac{|(S_{ij} \cup C_k) \in I|}{|S_{ij} \in A_i|} log \frac{|(S_{ij} \cup C_k) \in I|}{|S_{ij} \in A_i|} \right]$$

where;

I = set of i^{th} attribute value and i^{th} class value pairs from the dataset,

S_i = set of unique values that the i^{th} attribute can take,

C = set of unique classes in the dataset

Rule extraction starts with the minimum number of conditions. It can manually be set to any number ranging from 1 to n_a. If all the examples in the dataset cannot be classified by the rules extracted while satisfying the current number of conditions (Alg1-Ln6), it is incremented by one (Alg1-Ln7) and the process is repeated until there are no more unclassified examples left. If the number of conditions reaches the maximum value, which is equivalent to the number of attributes, then the remaining unclassified examples are all considered to be rules individually.

In order to minimize the number of combinations when the condition number is greater than one, SILEA follows an

approach similar to Sequential Forward Selection (Alg3). This eliminates consideration of every possible combination because in each iteration, SILEA fixes an attribute from previous iteration. It selects $(n_c - 1)$ number of attributes with least entropies where n_c=condition number (Alg3-Ln2-4). Then, it finds all the combinations of these pre-selected attribute(s) with the remaining attribute(s) by appending the remaining attributes one by one to the pre-selected ones (Alg3-Ln5-8). For example, to determine combinations when condition number is 3, the 2 attributes with least entropy values are selected and then the remaining attributes are appended one by one to form the combinations for rule extraction. Therefore, if the number of attributes within the example set were 5, the combinations to be considered for rule extraction would be {(Attr1, Attr2, Attr3); (Attr1, Attr2, Attr4); (Attr1, Attr2, Attr5)}. For each combination, the algorithm goes through the rule extraction mechanism employed by SILEA which is explained in detail in Section III-C.

C. Rule Extraction Procedure

For each selected combination of attributes, SILEA extracts every possible rule in a simple yet accurate way. The main idea is that the attribute combinations along with their classes for each example in the example set are considered to be potential rules unless a contradiction among examples occur. Contradiction is when an attribute combination belongs to different classes in the dataset. Two different sets are used throughout the extraction process, *Blacklist* (Alg1-Ln8) and *PotentialList* (Alg1-Ln9). The *Blacklist* contains attribute combinations which cannot form a rule and the *PotentialList* contains attribute combinations with their classes which are likely to be rules. Potential rules in *PotentialList* also contain the number of examples they can classify.

Initially, for each example (Alg1-Ln10), the attribute combination is checked by the algorithm whether or not it exists in the *Blacklist* (Alg1-Ln13). If it exists, then the current combination for the current example is ignored since it cannot form a rule and the next combination is considered. If the combination does not exist in the *Blacklist*, then the algorithm checks whether the combination exists in the *PotentialList* (Alg1-Ln14-22). There are three cases that can occur in this case. First case is that the combination does not exist in the *PotentialList* (Alg1-Ln14). Then, it is added to the *PotentialList* and its occurrence value is set to 1 (Alg1-Ln15-16). Second case is that it exists in the *PotentialList* and the rule combination has the same class as the one in the *PotentialList* (Alg1-Ln17). In this case, the current combination can still be considered as a potential rule, therefore it is kept in the *PotentialList* and its occurrence value is incremented by 1 (Alg1-Ln18). Third case is that it exists in the *PotentialList* and the combination has a different class than the one in the *PotentialList* (Alg1-Ln19). Then, the combination cannot form a rule, therefore it is removed from the *PotentialList* and placed in the *Blacklist* so that next time such combinations can be ignored (Alg1-Ln20-21). After every example is visited and processed in a single pass, potential rules within the *PotentialList* would stand to form rules.

Algorithm 4: ***Filter***(*PotentialList*, *UnclassifiedExamples*)

```
1 for each Rule ∈ PotentialList do
2     RuleClassifies = false;
3     for each Example ∈ UnclassifiedExamples do
4         if Classifies(Rule, Example) then
5             RuleClassifies = true;
6             UnclassifiedExamples =
                UnclassifiedExamples − Example;
7     if !Classifies(Rule, Example) then
8         PotentialList = PotentialList − Rule;
```

TABLE I
ALGORITHM COMPLEXITIES

(n_a = # of attributes, m_{PRSET} = # of expressions stored in PRSET)

Algorithm	Number of combinations	Asymptotic growth
SILEA	$\frac{1}{2}n_a(n_a+1)$	$O(n^2)$
RULES3-Plus	$n_a + m_{PRSET}\sum_{i=1}^{n_a-1} n_a - 1$	$O(n^3)$
RULES3	$\sum_{i=1}^{n_a} \frac{n_a!}{(n_a-i)!i!}$	$O(nn!)$

Examples are classified by the extracted rules in descending order of their occurence values (Alg4). If all the examples that a rule can classify are also classified by another rule with a higher occurence value, then the rule with lower occurence value is discarded since it becomes obsolete (Alg4-Ln9-11). The algorithm also discards the examples that can be classified by the selected rules (Alg4-Ln4-7). At the end of this filtering process, the rules in *PotentialList* are added to the *SelectedRules* list (Alg1-Ln27).

The feature selection complexity of SILEA along with some other algorithms are provided in Table I. SILEA's efficiency, especially in terms of the number of combinations that are needed to be considered, surpasses the algorithms it is compared to. In order to visualize the complexity of feature selection of the algorithms, let's assume an example set where $n_a = 15$. For this particular example set, the maximum number of combinations that each algorithm considers are: 32,766 for RULES3, 2,955 for RULES3-Plus and 120 for SILEA. The number of combinations considered in SILEA is considerably lower than both RULES3 and its successor RULES3-Plus algorithms.

IV. ILLUSTRATIVE PROBLEM

The Car Acceleration dataset [28] is used to illustrate the execution of SILEA algorithm. This particular dataset consists

TABLE II
EXAMPLE SET

Example	Fuel	Max-Speed	Car-Size	Acceleration
1	diesel	high	large	good
2	propane	high	large	good
3	petrol	high	compact	excellent
4	petrol	high	large	excellent
5	diesel	low	medium	good
6	petrol	low	compact	good
7	petrol	average	medium	excellent
8	diesel	average	medium	poor

TABLE III
PotentialList - POTENTIAL RULES EXTRACTED FROM THE 1ST EXAMPLE

n_c	#	Rules
1	1	$Fuel = diesel \rightarrow$ $Acceleration = good$
1	1	$Max\text{-}Speed = high \rightarrow$ $Acceleration = good$
1	1	$Car\text{-}Size = large \rightarrow$ $Acceleration = good$

of examples where the algorithm needs to execute all of its cases at some time during the extraction. This makes it easier to demonstrate how the algorithm functions in different cases. The dataset consists of three attributes and three classes. The attributes are; Fuel, Max-Speed and Car-Size. A combination of these attributes corresponds to a certain acceleration performance of the car which could be good, excellent or poor.

The example set is given in Table II where attributes are sorted according to their entropies. In this example set, Fuel has the lowest entropy while Car-Size has the highest. Quantization is not applied since there are no numerical attributes in the dataset. Minimum number of conditions gets incremented to 1.

Then, for each example in the dataset, the algorithm goes through the following steps:

For each example; the *FormedRules* list is formed and filled by the ***GenerateFormedRules*** function. This list contains the expressions generated from the example which are to be checked whether or not they form a rule. For the first example in the set, the following expressions are formed;

- $Fuel = diesel \rightarrow Acceleration = good$
- $Max\text{-}Speed = high \rightarrow Acceleration = good$
- $Car\text{-}Size = large \rightarrow Acceleration = good$

Since there are no items in both the *Blacklist* and the *PotentialList*, these combinations are added to the *PotentialList* and their occurrences are set to 1 for each one of them as shown in Table III;

In the second example, the following expressions are formed;

- $Fuel = propane \rightarrow Acceleration = good$
- $Max\text{-}Speed = high \rightarrow Acceleration = good$
- $Car\text{-}Size = large \rightarrow Acceleration = good$

TABLE IV
PotentialList - POTENTIAL RULES EXTRACTED FROM THE 1ST AND 2ND EXAMPLES

n_c	#	Rules
1	2	$Max\text{-}Speed = high \rightarrow$ $Acceleration = good$
1	2	$Car\text{-}Size = large \rightarrow$ $Acceleration = good$
1	1	$Fuel = diesel \rightarrow$ $Acceleration = good$
1	1	$Fuel = propane \rightarrow$ $Acceleration = good$

TABLE V
Blacklist - ATTRIBUTE NAME(S) WHICH BELONG TO MORE THAN ONE CLASS

n_c	Conditions
1	$Max\text{-}Speed = high$

TABLE VI
PotentialList - POTENTIAL RULES EXTRACTED FROM THE 1ST, 2ND AND 3RD EXAMPLES

n_c	#	Rules
1	2	$Car\text{-}Size = large \rightarrow$ $Acceleration = good$
1	1	$Fuel = diesel \rightarrow$ $Acceleration = good$
1	1	$Fuel = propane \rightarrow$ $Acceleration = good$
1	1	$Fuel = petrol \rightarrow$ $Acceleration = excellent$
1	1	$Car\text{-}Size = compact \rightarrow$ $Acceleration = excellent$

Since there are no items in the *Blacklist*, and there exist some combinations in the *PotentialList*, each one of the newly generated expressions are compared with those in the *PotentialList*. Since there are no potential rules containing the attributes of the first expression, it is added to the *PotentialList* and its occurence is set to 1. The second and the third expressions do exist in the *PotentialList* and since they have the same class, there exists no contradiction. Therefore, the rules are left in the *PotentialList* but their occurrence values are incremented. The list would be as shown in Table IV;

In the third example, the following expressions are formed;

- $Fuel = petrol \rightarrow Acceleration = excellent$
- $Max\text{-}Speed = high \rightarrow Acceleration = excellent$
- $Car\text{-}Size = compact \rightarrow Acceleration = excellent$

There exist no items in *Blacklist*, therefore, the expressions are compared with items in the *PotentialList*. The first and the third expressions do not contradict with any of the rules in the *PotentialList*, therefore they are added to it. The second one however exists in the *PotentialList* with a different class. So the contradicted rule; $Max\text{-}Speed = high \rightarrow Acceleration = good$, is removed from the *PotentialList* and the attribute of this rule is added to the *Blacklist* to be ignored next time it is observed. The lists are shown in Table V and Table VI;

The same process is applied to every example in the

TABLE VII
PotentialList - POTENTIAL RULES SELECTED FROM ALL THE EXAMPLES WITH $n_c = 1$ AND FROM THE 1ST EXAMPLE WITH $n_c = 2$

n_c	#	Rules
1	2	$Car\text{-}Size = low \rightarrow$ $Acceleration = good$
1	1	$Fuel = propane \rightarrow$ $Acceleration = good$
2	1	$Fuel = diesel \& Max\text{-}Speed = high \rightarrow$ $Acceleration = good$
2	1	$Fuel = diesel \& Car\text{-}Size = large \rightarrow$ $Acceleration = good$

example set. After visiting every example in the set, the *PotentialList* will contain all the rules with condition number n_c, which is currently set to 1.

After the rule extraction, ***Filter*** function removes unnecessary rules from the *PotentialList*. Unnecessary rules are those that might have become obsolete since the examples they are able to classify can be classified by other rule(s) with higher occurrence value(s). Starting from the most classifying rules in the *PotentialList*, rules are checked whether they can classify existing unclassified examples. If the rules can classify at least one example in the *UnclassifiedExamples* set, then the examples that the rules can classify are removed from the set and the rest of the rules are checked with the remaining unclassified examples. If the rules cannot classify the unclassified examples or if there are no more examples in the *UnclassifiedExamples* set, then these rules are removed from the *PotentialList*. After the execution of the ***Filter*** function, the remaining rules in the *PotentialList* are added to the *SelectedRules* list.

Since there are still remaining unclassified examples, the number of conditions is incremented, the first attribute is fixed since $(n_c - 1 = 1)$, and all the combinations with the remaining attributes are calculated. The following expressions are generated from the first example by the ***GenerateFormedRules*** function;

- $Fuel = diesel \& Max\text{-}Speed = high \rightarrow Acceleration = good$

- $Fuel = diesel \& Car\text{-}Size = large \rightarrow Acceleration = good$

Since there are no items in both the *Blacklist* and the *PotentialList*, these combinations are added to the *PotentialList* and their occurrences are set to 1 as shown in Table VII;

All the other examples are visited one by one going through the same steps as above. Rules in Table VIII are what are left in the *SelectedRules* list at the end of extraction and selection processes.

Since the rules in the *SelectedRules* list can classify every example in the dataset, the algorithm selects all the rules in the *SelectedRules* list and terminates.

V. EXPERIMENTAL RESULTS

In this section, we compare the performance of SILEA to some of the well-known algorithms in the inductive learning field using 5 different datasets. The datasets used for evaluation are; Balloons [15], Hayes-Roth [11], Hepatitis [10], Iris [9] and Lenses [4]. For each of the datasets, we generated 10 randomly sorted versions of the datasets and recorded their average performances along with the average number of rules each algorithm extracted for the given dataset. The dataset was split as 60% and 40% for training and testing, respectively.

Tables IX, X, XI, XII and XIII present a number of experiments and their performances. The number of conditions set for SILEA and RULES algorithms is equal to 1 for all the datasets and the quantization levels set for the Iris dataset is 3 and for the rest of the datasets is 5. RULES3-Plus requires an additional parameter to be set, which is called PRSET. For each dataset, this value was set equal to the number of attributes of the dataset's examples in order to assure the algorithm to generate its best performance results. The remaining algorithms were run with their default parameter settings as they are the ones that were suggested to be used.

TABLE VIII
SelectedList - ALL SELECTED RULES FROM THE DATASET

n_c	#	Rules
1	2	$Car\text{-}Size = low \rightarrow$ $Acceleration = good$
1	1	$Fuel = propane \rightarrow$ $Acceleration = good$
2	2	$Fuel = petrol \& Max\text{-}Speed = high \rightarrow$ $Acceleration = excellent$
2	1	$Fuel = diesel \& Max\text{-}Speed = average \rightarrow$ $Acceleration = poor$
2	1	$Fuel = diesel \& Max\text{-}Speed = high \rightarrow$ $Acceleration = good$
2	1	$Fuel = petrol \& Max\text{-}Speed = average \rightarrow$ $Acceleration = excellent$

TABLE IX
PERFORMANCES FOR THE BALLOONS DATASET

Algorithms	Avg. # of rules	μ	σ
SILEA	3	**100%**	0
RULES3	3	**100%**	0
RULES3-Plus	6	**100%**	0
C4.5	3	90.0%	16.1
CN2	3	**100%**	0
RIPPER	2	84.4%	21.08
RIDOR	2	71.1%	13.04
PART	3	90.0%	16.1
DecisionTable	4	86.7%	17.21
RandomTree	6	88.9%	18.89

TABLE X
PERFORMANCES FOR THE HAYES-ROTH DATASET

Algorithms	Avg. # of rules	μ	σ
SILEA	24	**84.1%**	3.92
RULES3	21	72.3%	4.5
RULES3-Plus	46	64.4%	3.68
C4.5	12	80.7%	5.4
CN2	18	70.9%	9.05
RIPPER	6	72.8%	8.05
RIDOR	7	68.0%	12.56
PART	9	78.3%	7.46
DecisionTable	6	55.0%	3.03
RandomTree	50	74.4%	8.81

TABLE XI
PERFORMANCES FOR THE HEPATITIS DATASET

Algorithms	Avg. # of rules	μ	σ
SILEA	23	**82.6%**	3.47
RULES3	33	73.3%	4.92
RULES3-Plus	41	73.7%	4.63
C4.5	5	78.2%	2.66
CN2	12	79.0%	2.4
RIPPER	3	79.0%	3.65
RIDOR	3	78.5%	4.44
PART	7	79.4%	3.63
DecisionTable	13	76.8%	2.43
RandomTree	48	76.6%	5.65

TABLE XII
PERFORMANCES FOR THE IRIS DATASET

Algorithms	Avg. # of rules	μ	σ
SILEA	5	**96.7%**	2.36
RULES3	5	86.1%	4.07
RULES3-Plus	19	**96.7%**	2.36
C4.5	4	93.5%	3.46
CN2	5	94.5%	3.41
RIPPER	4	92.3%	2.85
RIDOR	3	93.7%	3.41
PART	4	93.8%	4.01
DecisionTable	3	94.3%	2.96
RandomTree	12	94.8%	2

TABLE XIII
PERFORMANCES FOR THE LENSES DATASET

Algorithms	Avg. # of rules	μ	σ
SILEA	6	**80.0%**	13.33
RULES3	6	68.0%	15.49
RULES3-Plus	9	48.0%	14.76
C4.5	3	**80.0%**	12.47
CN2	4	70.0%	9.43
RIPPER	2	66.0%	5.16
RIDOR	2	60.0%	14.91
PART	3	**80.0%**	12.47
DecisionTable	2	62.0%	13.17
RandomTree	11	69.0%	18.53

Even though the number of attribute combinations to be considered are reduced in SILEA to O(n^2), it still was able to perform either as good or better than other algorithms on averages of all 5 cases as seen in Tables XIV. Considering average performances for all datasets, SILEA was 8.8% better than RULES3, 12.1% better than RULES3-Plus, 4.2% better than C4.5, 5.8% better than CN2, 9.8% better than RIPPER, 14.4% better than RIDOR, 4.4% better than PART, 13.7% better than DecisionTable and 8.0% better than RandomTree algorithms. Standard deviations show that the performance results of SILEA, in most cases, were either close to or more stable than the others.

Compared to the similar algorithms like the RULES algorithms, SILEA was able to achieve these performance results by reducing the number of rules in 4 out of 5 datasets. The difference in terms of the number of rules generated between SILEA and the other algorithms on average is very small.

TABLE XIV
AVERAGE PERFORMANCES

Algorithms	μ
SILEA	**88.7%**
RULES3	79.9%
RULES3-Plus	76.6%
C4.5	84.5%
CN2	82.9%
RIPPER	78.9%
RIDOR	74.3%
PART	84.3%
DecisionTable	75.0%
RandomTree	80.7%

This shows that SILEA was able to extract as general rules as possible for each iteration with these datasets. SILEA's method of rule selection among the generated rules also work accurately since its performance results have excelled the other algorithms with the analyzed datasets.

VI. CONCLUSION AND FUTURE WORK

SILEA is a simple, yet accurate inductive learning algorithm. It tries to minimize the enormous amount of possible consideration instances, i.e., O($n.n!$), to a reasonable amount, i.e., O(n^2), without sacrificing from its accuracy. The expected performance drop from minimizing the number of combination of attributes to be considered is recovered by two factors employed in the algorithm. First one is that the algorithm, for the given combination, extracts all the rules and selects those which can classify the most number of examples. During the selection phase, the algorithm also discards any rules that might have been set obsolete by other rules with higher classification capabilities. Second one is that since the algorithm favors certain attributes over the others when performing combination reduction, it is made sure that these biased attributes are those which excel the others based on their entropy values. SILEA, however, does not guarantee the most general rules for a given dataset. Even though the algorithm assures the generation of the most general rules for the considered combination, it is possible for it to miss more general rules regarding the overall extraction since it eliminates certain combinations in each iteration.

SILEA can be improved for different scenarios by incorporating other approaches. For instance, the algorithm can employ a better range calculation technique to handle continuous attributes in a more accurate way. Another improvement can be made on the rule extraction process by introduction of tolerance of error in the rules it generates. SILEA also lacks the ability to update the model it generates from a dataset. Every time new training data is to be used to train the model, the entire process is to be repeated along with the previous data. An incremental version of SILEA could help address this issue by letting users update their models without the need to re-generate rules from the data which had already been processed.

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under award number CNS-1321164 and IIA-1301726.

REFERENCES

[1] M. S. Aksoy, "A review of rules family of algorithms," *Mathematical and Computational Applications*, vol. 13, no. 1, p. 51–60, 2008.

[2] A. An, "Learning classification rules from data," *Computers & Mathematics with Applications*, vol. 45, no. 4, pp. 737–748, 2003.

[3] L. Burrell, O. Smart, G. K. Georgoulas, E. Marsh, and G. J. Vachtsevanos, "Evaluation of feature selection techniques for analysis of functional MRI and EEG," in *Proceedings of the International Conference on Data Mining (DMIN)*, 2007, pp. 256–262.

[4] J. Cendrowska, "UCI machine learning repository," 1990. [Online]. Available: http://archive.ics.uci.edu/ml

[5] J. Cheng, U. M. Fayyad, K. B. Irani, and Z. Qian, "Improved decision trees: a generalized version of ID3," in *Proceedings of the Fifth International Conference on Machine Learning (ICML)*, 1988, pp. 100–107.

[6] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine learning*, vol. 3, no. 4, pp. 261–283, 1989.

[7] H. ElGibreen and M. S. Aksoy, "RULES-TL: A simple and improved rules algorithm for incomplete and large data," *Journal of Theoretical and Applied Information Technology*, vol. 47, no. 1, pp. 28–40, 2013.

[8] H. Elgibreen and M. S. Aksoy, "RULES-IT: incremental transfer learning with rules family," *Frontiers of Computer Science*, vol. 8, no. 4, pp. 537–562, 2014.

[9] R. Fisher, "UCI machine learning repository," 1988. [Online]. Available: http://archive.ics.uci.edu/ml

[10] G. Gong, "UCI machine learning repository," 1988. [Online]. Available: http://archive.ics.uci.edu/ml

[11] B. . F. Hayes-Roth, "UCI machine learning repository," 1989. [Online]. Available: http://archive.ics.uci.edu/ml

[12] B. Hssina, A. Merbouha, H. Ezzikouri, and M. Erritali, "A comparative study of decision tree ID3 and C4.5," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 2, pp. 13–19, 2014.

[13] H. I. Mathkour, "RULES3-EXT: Improvements of rules3 induction algorithm," *Mathematical and Computational Applications*, vol. 15, no. 3, pp. 318–324, 2010.

[14] R. S. Michalski, "On the quasi-minimal solution of the general covering problem," in *Proceedings of the 5th international symposium on Information Processing (FCIP 69)*, vol. A3, pp. 125–128, 1969.

[15] M. Pazzani, "UCI machine learning repository," 1991. [Online]. Available: http://archive.ics.uci.edu/ml

[16] D. T. Pham and A. A. Afify, "RULES-6: a simple rule induction algorithm for supporting decision making," in *Proceedings of the 31st Annual Conference of IEEE Industrial Electronics Society, (IECON)*, Nov 2005, pp. 2184–2189.

[17] D. T. Pham and M. S. Aksoy, "A new algorithm for inductive learning," *Journal of Systems Engineering*, vol. 5, no. 2, pp. 115–122, 1995.

[18] D. T. Pham, S. Bigot, and S. S. Dimov, "RULES-5: a rule induction algorithm for classification problems involving continuous attributes," in *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 217, no. 12, pp. 1273–1286, 2003.

[19] ——, "RULES-F A fuzzy inductive learning algorithm," in *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 220, no. 9, pp. 1433–1447, 2006.

[20] D. T. Pham and S. S. Dimov, "An algorithm for incremental inductive learning," in *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 211, no. 3, pp. 239–249, 1997.

[21] D. T. Pham and M. S. Aksoy, "An algorithm for automatic rule induction," *Artificial Intelligence in Engineering*, vol. 8, no. 4, pp. 277–282, 1993.

[22] D. T. Pham and S. S. Dimov, "An efficient algorithm for automatic knowledge acquisition," *Pattern Recognition*, vol. 30, no. 7, pp. 1137–1143, 1997.

[23] J. R. Quinlan, "Generating production rules from decision trees," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, vol. 87, ser. IJCAI'87, pp. 304–307, 1987.

[24] ——, "Learning logical definitions from relations," *Machine learning*, vol. 5, no. 3, pp. 239–266, 1990.

[25] ——, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[26] J. Quinlan, "Learning efficient classification procedures and their application to chess end games,", vol. 1, pp. 463–482, 1983.

[27] A. W. Whitney, "A direct method of nonparametric measurement selection," *IEEE Transactions on Computers*, vol. 20, no. 9, pp. 1100–1103, 1971.

[28] R. Yasdi, "Learning classification rules from database in the context of knowledge acquisition and representation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 3, pp. 293–306, 1991.