On the Appropriateness of Evolutionary Rule Learning Algorithms for Malware Detection

M. Zubair Shafiq, S. Momina Tabish, Muddassar Farooq

Next Generation Intelligent Networks Research Center (nexGIN RC)
FAST National University of Computer & Emerging Sciences (NUCES)
Islamabad, 44000, Pakistan
{zubair.shafiq,momina.tabish,muddassar.farooq}@nexginrc.org

ABSTRACT

In this paper, we evaluate the performance of ten well-known evolutionary and non-evolutionary *rule learning algorithms*. The comparative study is performed on a real-world classification problem of detecting malicious executables. The executable dataset, used in this study, consists of 189 attributes which are statically extracted from the executables of Microsoft Windows operating system. In our study, we compare the performance of rule learning algorithms with respect to four metrics: (1) classification accuracy, (2) the number of rules in the developed rule set, (3) the comprehensibility of the generated rules, and (4) the processing overhead of the rule learning process. The results of our comparative study suggest that evolutionary rule learning classifiers cannot be deployed in real-world malware detection systems.

Categories and Subject Descriptors

D.4.6 [**Software**]: Security and Protection—*Invasive software*; I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning, Induction*

General Terms

Algorithms, Experimentation, Security

Keywords

Genetics Based Machine Learning, Learning Classifier Systems, Malware Detection

1. INTRODUCTION

The genetic based machine learning systems, more commonly known as GBML, have become significantly more sophisticated compared with primitive versions of the learning classifier systems (LCS) proposed by John Holland in 1970s. A number of LCS variants exist that include evolutionary rule learners, evolutionary neural networks and other hybrid schemes. But evolutionary rule learners have received significantly more attention compared with other LCS. They can be further subdivided into Michigan-style, Pittsburgh-style, anticipatory, and iterative rule learners [1]. Most of the recent work in LCSs is focused on classical Michigan- and Pittsburgh-style evolutionary rule learners, which include but are not limited to XCS, UCS, GAssist and GALE.

Michigan- and Pittsburgh-style rule learners are utilized in a number of real-world problems such as data mining, classification and optimization [13]. In data mining and classification, both Michigan- and Pittsburgh-style rule learners have shown competitive performance compared with non-evolutionary algorithms. Recent studies such as [13], [14], [15] and [16] have evaluated them on a number of benchmark problems (such as UCI machine learning repository [21]) and have reported promising classification results. However, most of these studies are limited to evaluating the classification accuracy and the algorithmic behavior.

In this paper, we evaluate several well-known evolutionary rule learning classification algorithms using a real-world classification problem of malware detection. The malware detection problem demands—in addition to the classification accuracy—a number of additional requirements: (1) processing overheads to ensure the feasibility of realtime deployment, and (2) comprehensibility of the developed solution that helps malware forensic experts. Therefore, malware detection problem is relatively more challenging because we need to simultaneously optimize accuracy and efficiency of classification.

The rule learning classification algorithms can be broadly divided into two major categories: (1) evolutionary rule learners, and (2) non-evolutionary rule learners. In our comparative study, we select five well-known algorithms from each category. The choice of five evolutionary classifiers is as following: two state-of-the-art Michigan-style classifiers: XCS [2] and UCS [3], two state-of-the-art Pittsburgh-style classifiers: GAssist-ADI [4] and GAssist-Intervalar [5], and a genetic fuzzy classifier SLAVE [6]. The choice of five non-evolutionary classifiers is as following: three well-known iterative rule learners: RIPPER [7], SLIPPER [8] and PART [9], a decision tree based rule learner C4.5 rules [10], and an instance based rule learner RIONA [11].

In order to undertake a comprehensive study, we use four performance metrics for comparison: (1) classification accuracy, (2) the number of rules, (3) comprehensibility of the rules, and (4) processing overhead. To factor out implemen-

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

tation related bias in our study, we use the implementations of rule learning algorithms provided by a unified framework called Knowledge Extraction based on Evolutionary Learning (KEEL) [20].

The remaining paper is organized as follows. We provide details of the executable dataset in Section 2. We explain the results of our experimental study in Section 3. We analyze the results of four performance metrics separately. In Section 4 we present an overview of the related work in which evolutionary rule learning algorithms are compared with non-evolutionary classification algorithms. Finally, we conclude our paper in Section 5 and provide some valuable insights obtained from our study.

2. EXECUTABLE DATASET

In this section, we present an overview of the executable dataset used in our study. The executable dataset consists of 11,786 executable files for the Microsoft Windows operating system in portable executable format [27]. A host of features—189 to be precise—are statically extracted from each executable. The executable dataset consists of two types of executables: *benign* and *malicious*.

2.1 Benign Executables

We have collected 1,447 benign PE files from the local area network of our virology lab. The collection contains executables ranging from Packet CAPture (PCAP) file parsers compiled by MS Visual Studio 6.0 to MS Windows XP/Vista applications' executables.

2.2 Malicious Executables

We have obtained 10,339 malicious executables from a publicly available malware database called 'VX Heavens Virus Collection' [30]. The malicious executables are subdivided into eight major categories such as *virus*, *trojan*, *worm*, etc. Moreover, we have combined some categories that have similar functionality. For example, we have combined *constructor* and *virtool* to create a single *constructor + virtool* category. This unification increases the number of malware samples per category. We now provide a brief introduction of each category of malware below in order to make the paper self contained.

2.2.1 Backdoor + Sniffer

A backdoor is a program which allows bypassing of standard authentication and verification methods of an operating system. As a result, the remote access to a computer system is possible without an explicit consent of the user [28]. The information logging and sniffing activities are easily made possible because of the gained remote access. One of the important backdoors used in our study are `Backdoor.Win32.IRCBot` and its different variants. This backdoor proliferates using the well-known instant messaging clients and effectively makes the compromised host a botnet client. We have combined *backdoor* and *sniffer* categories because of their similar functional behavior.

2.2.2 Constructor + Virtool

This category of malware mostly includes toolkits for automatically creating new malware by varying a given set of input parameters [28]. `Constructor.Win32.SennaSpy` and its variants are well-known examples of malware in this category. The `Constructor.Win32.SennaSpy` generates "customized" trojans that allow remote access. *Virtool* and *constructor* categories are combined because of their similar functionality.

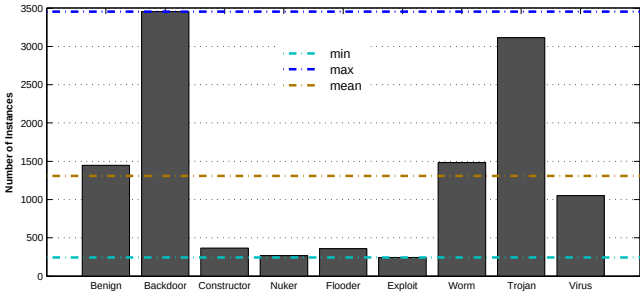

Figure 1: Class distribution of the executable dataset used in this study

2.2.3 DoS + Nuker

Both *DoS* and *nuker* based malware allow an attacker to launch malicious activities at a victim's computer system that can possibly result in the denial of service (DoS) attack. These activities can result in slow down, restart, crash or shutdown of a computer system [28]. `Nuker.Win32.WinNuke` and its variants are well-known examples of malware in this category. `Nuker.Win32.WinNuke` affects older versions of Windows and causes them to crash.

2.2.4 Email- + IM- + SMS Flooder

The malware in this category initiate unwanted information floods such as email, instant messaging and sms floods [28]. A popular malware of this category is `IM-Flooder.Win32.CriminalMSN`, which slows down the computer system, posts advertisement popups and sends spam emails from the mailbox.

2.2.5 Exploit + Hacktool

The malware in this category exploit vulnerabilities in the software implementations which, most commonly, result in buffer overflows [28]. A representative sample of this malware category is `Exploit.Win32.SQLexp.c`, which launches well-known SQL injection attacks.

2.2.6 Email- + IM- + IRC- + Net Worm

The malware in this category spread through instant messaging networks, IRC networks and port scanning [28]. `Email-Worm.Win32.Mydoom` and its variants are popular malware in this category. This worm spreads using emails and popular P2P networks.

2.2.7 Trojan

A trojan is a broad term that refers to stand alone programs which appear to perform a legitimate function but covertly do unwanted, and possibly harmful, activities such as providing remote access, data destruction and corruption [28]. A typical example of *trojan* category is `Trojan.Win32.Icekboy`, which downloads other malware and also disables key functions of Windows.

2.2.8 Virus

A virus is a program that can replicate itself and attach itself with other benign programs. It is probably the most

well-known type of malware [28]. `Virus.Win32.Bolzano` and its variants are examples of malware in this category. It not only infects other executable files but also denies the user the right to modify any file on the file system.

For testing purposes, eight separate datasets are created by combining *benign* executables with each category of the *malicious* executables. Figure 1 shows the class distribution of the executable dataset used in our study.

3. RESULTS AND DISCUSSIONS

In this section, we provide the results of our experiments and relevant discussions. We have used the implementations of the rule learning algorithms present in a publicly available framework, called KEEL (Knowledge Extraction based on Evolutionary Learning) [20]. We have used the default parameters of all algorithms as provided in KEEL, unless stated otherwise. We compare the performance of classifiers on four performance metrics: (1) the classification accuracy, (2) the number of rules, (3) the comprehensibility of rules, and (4) the processing overheads. Each of the above-mentioned metrics is discussed separately below.

3.1 Classification Accuracy

In a typical two-class problem, such as malicious executable detection, the classification decision of an algorithm may fall into one of the following four categories:

1. **true positive** (TP); correct classification of a malicious executable as malicious.
2. **true negative** (TN); correct classification of a benign executable as benign.
3. **false positive** (FP); wrong classification of a benign executable as malicious.
4. **false negative** (FN); wrong classification of a malicious executable as benign.

A suitable metric for quantifying classification accuracy of an algorithm, also used in this study, is *accuracy*[1] which is defined as:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

We believe that it is important to provide some insights about the general characteristic indicators of our datasets as it would help a reader in appreciating the merits or demerits of different rule based classification algorithms. In our evaluation, we specifically focus on the 'quality of attributes' across all eight datasets. Several information-theoretic measures are proposed in the literature to evaluate the quality of attributes in a dataset. Information gain, gain ratio and symmetric uncertainty are to name a few [29]. These quality measures are often utilized for attribute selection.

Information gain (I) measures the reduction in uncertainty if the values of an attribute are known. For a given attribute X and a class attribute Y, the uncertainty is given by their respective entropies $H(X)$ and $H(Y)$. Then the information gain of X with respect to Y is given by $I(Y;X)$, where

[1]The terms *accuracy* and *classification accuracy* are interchangeably used in this paper.

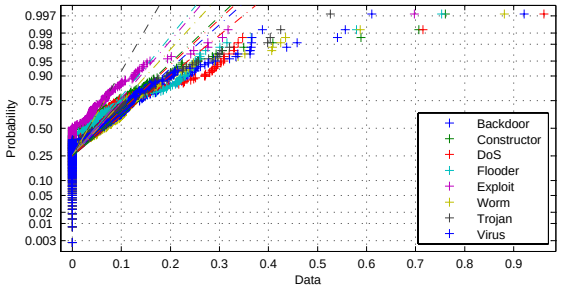

Figure 2: Normal probability plot for gain ratio of attributes

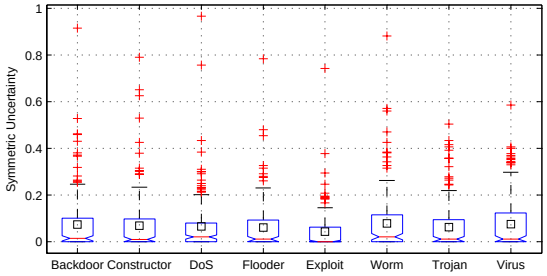

Figure 3: Box plot for distribution of symmetric uncertainty of attributes

$$I(Y;X) = H(Y) - H(Y|X)$$

However, a well-known problem with the information gain is its bias towards attributes split in multiple classes [29]. This limitation of information gain is overcome by another measure called gain ratio (GR). The gain ratio of an attribute X with respect to a class attribute Y is given by $GR(Y;X)$, where

$$GR(Y;X) = \frac{I(Y;X)}{H(X)} = \frac{H(Y) - H(Y|X)}{H(X)}$$

The values of gain ratio are in the range of $[0, 1]$. The values approaching one represent features with higher classification power and vice-versa. We performed an analysis of the attributes' gain ratio distribution for all datasets used in this study. Figure 2 shows the normal probability plot for gain ratio of attributes of all datasets. Figure 2 clearly highlights the 'skewed' nature of the attributes' gain ratio distribution. This trend is consistent across all datasets. The majority of features have very low gain ratio; therefore, they can be marked as redundant. On the other hand, only a limited number of features have gain ratios greater than $\frac{1}{2}$, and hence they can prove valuable for achieving high classification accuracy.

It is also important to analyze the relative difficulty of datasets for classification. To perform this ranking we use another variant of information gain called symmetric uncertainty. Symmetric uncertainty is the combination of gain ratios, both for the given attribute with respect to a class attribute and vice-versa. This alleviates the non-symmetric nature of information gain which is a major limitation [29].

Table 1: Classification accuracy of all algorithms used in this study

Classifier	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	Average
Evolutionary Rule Based Algorithms									
XCS	0.9538± 0.0067	0.9607± 0.0045	0.9548± 0.0049	0.9555± 0.0047	0.9562± 0.0053	0.9528± 0.0083	0.9529± 0.0069	0.9532± 0.0043	0.9550± 0.0057
UCS	0.9833± 0.0027	0.9815± 0.0041	0.9840± 0.0040	0.9855± 0.0036	0.9840± 0.0038	0.9862± 0.0033	0.9862± 0.0028	0.9873± 0.0032	0.9847± 0.0034
GAssist-ADI	**0.9920± 0.0013**	**0.9931± 0.0012**	**0.9934± 0.0011**	**0.9927± 0.0011**	**0.9931± 0.0012**	**0.9927± 0.0011**	**0.9920± 0.0011**	**0.9942± 0.0022**	**0.9929± 0.0013**
GAssist-Intervalar	0.9877± 0.0026	0.9895± 0.0017	0.9859± 0.0037	0.9862± 0.0034	0.9866± 0.0021	0.9877± 0.0018	0.9855± 0.0018	0.9891± 0.0025	0.9873± 0.0024
SLAVE	**0.9920± 0.0013**	0.9920± 0.0004	0.9913± 0.0015	0.9920± 0.0016	0.9924± 0.0017	0.9924± 0.0016	0.9913± 0.0008	0.9906± 0.0015	0.9917± 0.0013
Non-Evolutionary Rule Based Algorithms									
RIPPER	0.9967± 0.0026	0.9964± 0.0038	0.9945± 0.0035	0.9963± 0.0024	**0.9971± 0.0037**	0.9949± 0.0039	0.9971± 0.0037	0.9956± 0.0028	0.9961± 0.0033
SLIPPER	**0.9982± 0.0019**	0.9971± 0.0022	**0.9964± 0.0029**	**0.9978± 0.0025**	0.9967± 0.0026	**0.9982± 0.0025**	**0.9982± 0.0019**	**0.9974± 0.0024**	**0.9975± 0.0023**
PART	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028	0.9956± 0.0028
C4.5 rules	0.9960± 0.0046	**0.9977± 0.0007**	0.9960± 0.0046	0.9960± 0.0046	0.9945± 0.0007	0.9977± 0.0007	0.9960± 0.0046	0.9960± 0.0046	0.9962± 0.0031
RIONA	0.9891± 0.0054	0.9891± 0.0054	0.9891± 0.0054	0.9891± 0.0054	0.9891± 0.0054	0.9922± 0.0047	0.9891± 0.0054	0.9922± 0.0047	0.9899± 0.0052
Average	$^{(6,7)}$ 0.9884 ±0.0032	$^{(1)}$ 0.9893 ±0.0027	$^{(8)}$ 0.9881 ±0.0034	$^{(4)}$ 0.9887 ±0.0032	$^{(5)}$ 0.9885 ±0.0029	$^{(3)}$ 0.9890 ±0.0031	$^{(6,7)}$ 0.9884 ±0.0032	$^{(2)}$ 0.9891 ±0.0031	0.9887 ±0.0031

Symmetric uncertainty is denoted by $U(X, Y)$, where

$$U(X,Y) = 2\left(\frac{I(Y;X)}{H(X)+H(Y)}\right) = 2\left(\frac{H(Y)-H(Y|X)}{H(X)+H(Y)}\right)$$

Figure 3 shows the box plots for the attributes' symmetric uncertainty distribution for all datasets. The black square boxes in the figure represent the mean values of symmetric uncertainty. The red plus signs represent the outliers that are outside 1st and 3rd quartiles. The black dotted horizontal lines indicate the median values of symmetric uncertainty. The *virus* dataset has the highest mean and median values for symmetric uncertainty; therefore, we can expect that it will have relatively higher classification accuracy. Similarly *exploit* dataset has the lowest mean and median values for symmetric uncertainty; therefore, we can expect that it will be difficult to classify it accurately.

Table 1 shows the classification accuracy of all (evolutionary and non-evolutionary) algorithms. We represent the best evolutionary and non-evolutionary classifier with a bold value for each category of malware. The best evolutionary rule learning algorithm is GAssit-ADI, which consistently outperforms other evolutionary algorithms for different categories of malware. It is closely followed by SLAVE which also has the average accuracy of more than 0.99. GAssist-Intervalar and UCS have accuracies in the range of approximately 0.98. XCS provides the worst accuracy amongst all classification algorithms used in our study. SLIPPER outperforms most of the non-evolutionary rule learning algorithms for different categories of malware. It is closely followed by C4.5 rules, RIPPER and PART respectively. The relatively worst classifier among non-evolutionary algorithm is RIONA that provides an accuracy of approximately 0.99.

It is evident from Table 1 that non-evolutionary rule learning algorithms clearly outperform their evolutionary counterparts for each category of malware. In fact, the best performing evolutionary rule learning algorithm, GAssist-ADI, provides lower accuracy compared with all non-evolutionary algorithms except RIONA.

Table 2: Number of rules in the finalized training model

Algorithm	Number of rules
XCS	$10,000.0 \pm 0.0$
UCS	$9,652.6 \pm 649.3$
GAssist-ADI	2.6 ± 0.7
GAssist-Intervalar	4.2 ± 0.7
SLAVE	3.3 ± 1.5
RIPPER	6.0 ± 0.8
SLIPPER	57.0 ± 3.0
PART	6.3 ± 0.7
C4.5 rules	7.4 ± 1.1
RIONA	-

The accuracy results of all algorithms, across different categories of malware, are approximately consistent with our expectation based on the distribution of symmetric uncertainty shown in Figure 3. *DoS* and *exploit* appear to be the most challenging malware types. On the other hand, *worm* and *virus* are relatively easier to classify. Superscripts in the last row of Table 1 show the relative ranking of datasets with respect to average classification accuracy.

3.2 Number of Rules

The number of rules used by an algorithm are directly dependent on input parameters. Sometimes, the maximum possible number of rules is explicitly limited via an input parameter. For all evolutionary algorithms, in this study, the maximum number of rules are limited by a *MaxPopSize* parameter. This parameter is set to $10,000$ for XCS and UCS, 400 for GAssist-ADI and GAssist-Intervalar, and 100 for SLAVE. No algorithm, except for XCS and UCS, reached its respective upper limit during the experiments. In comparison, no non-evolutionary rule learning algorithms (used in our study) explicitly puts a limit on the number of rules.

Table 2 shows that the number of rules in XCS and UCS are limited by the *MaxPopSize* ($N = 10,000$) parameter as

they try to over-fit.[2] In comparison, other evolutionary rule learning algorithms utilize significantly fewer rules for classifying malware. Non-evolutionary rule learning algorithms also generate significantly less number of rules. However, SLIPPER appears to be the only exception, which generates more than fifty rules on the average. No results are reported for RIONA in Table 2 because it does not create a global rule set like other rule learning algorithms used in this study. Refer to [11] to get a better understanding of this process in RIONA.

3.3 Comprehensibility of Rules

The comprehensibility of generated rules is another important metric that is used for quantifying the performance of different rule learning algorithms. It is important to highlight that the comprehensibility of rules provide great help to the 'malware forensic experts' in understanding the behavior and characteristics of a given malware. The comprehensibility of rules is a subjective term and it is difficult to precisely and accurately quantify it. However, we believe that the principle of "less precise, more accurate" is also applicable here. In order to make our analysis more accurate, we quantify the comprehensibility of rules by using coarse grain Low, Medium, and High categories. It should be noted that we consider the legibility of the individual rules and the size of rule set to establish the comprehensibility of rules.

A few samples of the rules generated by all algorithms are shown in Table 3. Both XCS and UCS develop conjunctive IF-THEN rules containing specified intervals for *every* attribute, i.e., the size of rule antecedent is maximum. This factor is important here because the dimensionality of the dataset used in this paper (= 189) is quite high. The sizes of the individual rules developed by XCS and UCS are extremely large so their legibility is very poor. Moreover, the number of rules in the final rule set are very large compared with other algorithms (see Table 2). It should be noted that the rules are maintained in the form of *decision-list* for the classification phase [22], consequently they are context- and order-dependent. They become hard to interpret as size of the rule set increases. Due to these obvious shortcomings, we say that the comprehensibility of the rules generated by XCS and UCS is *Low*.

GAssist-ADI mostly develops rules by using single-sided ranges and also has *default* rules. It is noteworthy that GAssist-ADI does not use all input attributes in rule antecedents. Also the size of the rule set is very small. Therefore, the comprehensibility of rules generated by GAssist-ADI can be categorized as *High*. GAssist-Intervalar, similar to XCS and UCS, generates rules that contain specified intervals for *every* attribute (i.e., the size of rule antecedent is the maximum). However, the final rule set contains only a few rules. So the comprehensibility of the generated rule set is categorized as *Medium*.

SLAVE uses fixed intervals (such as L1,L2,L3,L4) to generate fuzzy rules. It also assigns weights (usually rule fitness) to the individual rules. The use of weights makes it difficult to comprehend the rules. Moreover, the size of rule set is also very small. Overall, the comprehensibility of the rules generated by SLAVE can be ranked as *Medium*.

To conclude, non-evolutionary rule learning algorithms generate relatively less number of simpler rules. Therefore, the comprehensibility of the rules generated by RIPPER, SLIPPER, PART and C4.5 rules can be categorized as *High*. However, we cannot categorize RIONA because of similar reasons mentioned in Section 3.2.

[2]XCS's parameters are as follows: $N = 10,000, \alpha = 0.1, \beta = 0.2, \delta = 0.1, \nu = 10.0, \theta_{del} = 50.0, \epsilon_0 = 1.0$, GA Subsumption is applied with $\theta_{sub} = 50.0$. UCS's parameters that are shared with XCS have same values as indicated. For UCS $acc_0 = 0.99$.

Table 4: Processing overheads for all algorithms used in this study

Algorithm	Execution time (sec)
XCS	109.0 ± 3.1
UCS	16.3 ± 2.0
GAssist-ADI	$1,177.2 \pm 245.2$
GAssist-Intervalar	$4,060.9 \pm 311.9$
SLAVE	775.1 ± 205.4
RIPPER	8.0 ± 0.7
SLIPPER	29.4 ± 13.6
PART	51.8 ± 4.8
C4.5 rules	37.0 ± 2.5
RIONA	2680.5 ± 578.4

3.4 Processing Overheads

Table 4 shows the processing overheads of all classification algorithms used in our study. The processing overheads are reported in Table 4 as the average time (in seconds) required for both training and testing per fold of the 10-fold cross validation process. It is important to include *both training and testing overheads* from a malware expert's point-of-view.

It is evident from Table 4 that the processing overheads of all evolutionary rule learning algorithms, except UCS and XCS, are significantly large compared with non-evolutionary classifiers. Malware detection systems are deployed in real operating systems; therefore, the processing overhead is a crucial metric. Note that GAssist-ADI and GAssist-Intervalar have relatively higher accuracies amongst evolutionary rule learning algorithms. But, very large processing overheads make them infeasible for realtime deployment in operating systems. The processing overhead of SLAVE is also not feasible for realtime deployment. UCS is the only evolutionary rule learning algorithm which has a processing overhead comparable with non-evolutionary algorithms.

All non-evolutionary rule learning algorithms, except RIONA, have acceptable processing overheads. The processing overhead of RIONA is large because it uses a combination of instance based learning and rule induction. For every given test instance, it first selects the neighboring instances similar to the nearest neighbor algorithm. After selection, it develops rule sets through rule induction. The complexity incurred due to exhaustive search of the neighboring instances for every test instance is a major performance bottleneck.

4. RELATED WORK

In this section, we present an overview of the related work in which evolutionary rule learning classification algorithms are compared with different non-evolutionary classification algorithms.

In [12], the authors compared the behavior of six competitive GBML approaches (UCS, GAssist, HIDER, HMOF, SLAVE and Fuzzy LogitBoost) with respect to six well-known machine learning techniques. The authors compared accuracy, number and comprehensibility of the developed rules for all algorithms using 20 datasets from the UCI repos-

Table 3: Rules developed by all algorithms used in this study

(a) XCS

```
IF MSPST32.DLL [0.0,0.3] ∧ MSFS32.DLL [0.0,0.3] ∧ ...  NumVersioninformation [0.0,0.7] ∧ NumUserdefined [0.0,0.8] :  malicious
IF MSPST32.DLL [0.0,0.2] ∧ MSFS32.DLL [0.0,0.7] ∧ ...  NumVersioninformation [0.0,0.8] ∧ NumUserdefined [0.4,1.0] :  benign
IF MSPST32.DLL [0.0,0.9] ∧ MSFS32.DLL [0.0,0.2] ∧ ...  NumVersioninformation [0.0,0.5] ∧ NumUserdefined [0.1,1.0] :  benign
```

(b) UCS

```
IF MSPST32.DLL [0.0,1.0] ∧ MSFS32.DLL [0.0,1.0] ∧ ...  NumVersioninformation [0.2,1.0] ∧ NumUserdefined [0.0,1.0] :  malicious
IF MSPST32.DLL [0.0,1.0] ∧ MSFS32.DLL [0.0,1.0] ∧ ...  NumVersioninformation [0.8,1.0] ∧ NumUserdefined [0.0,1.0] :  benign
IF MSPST32.DLL [0.0,1.0] ∧ MSFS32.DLL [0.0,0.6] ∧ ...  NumVersioninformation [0.5,1.0] ∧ NumUserdefined [0.0,1.0] :  benign
```

(c) GAssist-ADI

```
IF BHNETB.DLL [<0.8] ∧ NETSETUP.DLL [<0.5][>0.8] ∧ RPCTLS6.DLL [<0.6] ∧ Characteristics [>25851.3] ∧ CheckSum [<1.6E7] ∧ DllCharac-
teristics [<24622.5] ∧ RVAResourceTable [<5.8E7][>7.2E7] ∧ RVAExceptionTable [<111957.3] ∧ SizeCertificateTable [<1489.1]∧ RVADebug
[<7929568.0] ∧ RVAArchitecture [<710251.2][>1420502.4] ∧ SizeIAT [<2212.4] ∧ NumberofNameEntries [<35570.7] ∧ NumMessageTable [<0.5]
∧ NumUserdefined [<1.5] :  malicious
ELSE : benign
```

(d) GAssist-Intervalar

```
IF MSPST32.DLL [0.0,1.0] ∧ MSFS32.DLL [0.0,1.0] ∧ ...  NumVersioninformation [0.0,1.0] ∧ NumUserdefined [0.0,1.2] :  malicious
ELSE IF MSPST32.DLL [0.0,1.0] ∧ MSFS32.DLL [0.0,1.0] ∧ ...  NumVersioninformation [0.0,1.0] ∧ NumUserdefined [0.0,1.4] :  malicious
ELSE : benign
```

(e) SLAVE

```
IF RVAExportTable [L1,L2,L4] :  benign (W=0.8)
IF NumDialog [L2,L3] :  benign (W=1.0)
IF CCTN20.DLL [L1,L4] ∧ GDI32.DLL [L1,L2,L3,L4] ∧ NumAccelerators [L0,L1,L2] :  malicious (W=0.7)
```

(f) RIPPER

```
IF WSOCK32.DLL [>0.0] ∧ NumMessageTable [<=0.0] ∧ TimeDateStamp [<=1.015670976E9] :  malicious
ELSE IF dataSizeOfRawData [<=0.0] ∧ SHELL32.DLL [>0.0] malicious
ELSE : benign
```

(g) SLIPPER

```
IF TimeDateStamp [>9.9] :  malicious (W=0.7)
IF WS2_32.DLL [>0.0] ∧ MajorLinkerVersion [<=6.0] ∧ NumMessageTable [<=0.0] :  malicious (W=3.3)
ELSE benign (W=14.4)
```

(h) PART

```
IF MajorOperatingSystemVersion [>1.0] ∧ SizeOfInitializedData [>7680.0] :  benign
ELSE IF NumberofIDEntries [>1.0] :  malicious
ELSE : benign
```

(i) C4.5 Rules

```
IF NumMessageTable [<=0.0] ∧ SizeDebug [<=1.0] ∧ NumAccelerators [>0.0] ∧ NumCursor [<=7.0] :  malicious
ELSE IF NumMessageTable [<=0.0] ∧ SizeOfUninitializedData [>172032.0] :  malicious
ELSE : benign
```

itory [21]. The complexity of the rule learning process is also discussed in a subjective manner. They conclude that GBML techniques provide competitive accuracy while providing more interpretable models, as compared to well-known machine learning techniques.

In [13], the authors compared a Michigan-style classifier (XCS) with a Pittsburgh-style classifier (GAssist) using a number of datasets from the UCI repository [21]. In their experiments, both XCS and GAssist provided comparable classification accuracy. The authors conclude that small differences in their accuracies are attributable to the properties of the datasets.

In [14], the authors have compared a Michigan-style classifier (XCS) with a Pittsburgh-style classifier (GALE) and six non-evolutionary classifiers. They have conducted experiments on using 12 datasets from the UCI repository and other sources. They used ten fold cross validation technique in their evaluation. The results of their experiments show that both XCS and GALE provide significantly better classification accuracy compared with ZeroR, IB1 (k-nearest neighbor with $k = 1$), Naïve Bayes and PART. But both XCS and GALE have achieved comparable classification accuracies. Moreover, the classification accuracies of XCS and GALE are similar to those of IBk, C4.5 and SMO. The authors, however, have not analyzed the size and the comprehensibility of the developed solutions.

In [15], the authors have compared XCS to Bayesian network, SMO and C4.5 for classification of breast cancer data. The results of their experiments show that the classification accuracies of XCS and C4.5 are significantly better than those of Bayesian network and SMO. The authors have also performed rule compaction and analyzed the domain knowledge discovered by XCS and C4.5.

In [16], the authors compared XCS and UCS to C4.5, SMO and IB5 using 25 datasets. As per their analysis, XCS is the most robust algorithm, followed by IB5, UCS, C4.5 and SMO respectively. The authors argue that poor accuracy of C4.5 algorithm is because of specific problems in the datasets. The authors have also analyzed the impact of re-sampling techniques on the classification accuracy of algorithms. Their results show that the re-sampling techniques have generally improved the accuracies of algorithms. (Remember each algorithm has a different suitable re-sampling technique.)

In [17], the authors have evaluated XCS and UCS for intrusion detection using KDD Cup 1999 dataset. The results of their experiments show that XCS, in general, outperforms UCS. The authors have also performed a comparison with ten non-evolutionary classification algorithms. Their experiments show that the overall accuracy of XCS and UCS is significantly higher compared with non-evolutionary classifiers. The authors, however, have not studied the processing overheads of different algorithms, which is an important parameter for intrusion detection systems.

In [18], the authors have proposed an evolutionary classifier (EvoC) for medical diagnosis. They have also compared their proposed algorithm (EvoC) with three non-evolutionary algorithms, C4.5, PART and Naïve Bayes. They have used three different datasets in their study. Their results show that EvoC provides comparable accuracy as that of non-evolutionary algorithms. In [19], the authors have proposed to use ensemble of LCSs using bagging to improve classi-

fication accuracy in medical datasets. The proposed LCSE (Learning Classifier System Ensemble) not only outperforms a stand alone LCS but also non-evolutionary classifiers.

Contrary to the above-mentioned studies, in this paper we focus on the comparison of evolutionary *rule learning* classifiers with non-evolutionary rule learning classifiers. Specifically we evaluate different relevant metrics for rule based classifiers that include classification accuracy, the number of rules, the comprehensibility of the generated rules and the processing overhead of the rule learning process. To the best of our knowledge, this is the first comparative study of evolutionary rule learning algorithms for malware detection—an emerging security threat of the new millennium.

5. CONCLUSIONS AND INSIGHTS

In this paper, we have compared evolutionary rule learning algorithms with non-evolutionary rule learning algorithms for a real-world problem of executable malware detection. We have compared five well known evolutionary classifiers—XCS, UCS, GAssist-ADI, GAssist-Intervalar, SLAVE—with five non-evolutionary classifers—RIPPER, SLIPPER, PART, C4.5 rules, RIONA—using four performance metrics: (1) classification accuracy, (2) number of rules, (3) comprehensibility of the rules, and (4) processing overheads.

The results of our study show that non-evolutionary rule learning algorithms clearly outperform evolutionary rule learning algorithms for every performance metric. Table 5 provides a summarized comparison of all algorithms. We conclude that RIPPER, SLIPPER, PART and C4.5 rules consistently provide best performance with respect to all four performance metrics. In comparison, GAssist-ADI and SLAVE provide acceptable performance on all metrics except the processing overhead.

We also believe that the processing overheads and the comprehensibility of evolutionary rule learning algorithms can be significantly improved by integrating some concepts from non-evolutionary rule learning algorithms. The attributes should be ranked based on the global quality measures such as gain ratio and symmetric uncertainty. The higher ranking attributes can then be prioritized in the rule learning process. This will not only improve the comprehensibility of the rules but also reduce the number of rules in the final rule set. Moreover, it will also reduce the complexity of the rule learning process as well. It is important to emphasize that several techniques are proposed in the literature [23], [24], [25], [26] to reduce the rule's length and the size of rule set—ultimately leading to lower processing overheads. The authors of [15] have already proposed the idea of *rule compaction* to enhance the comprehensibility of the rule set generated by XCS. We believe that such kind of improvements should be incorporated into the core of evolutionary rule learning algorithms in order to make them competitive in realtime classification systems. These efforts, though highly desirable, are not within the scope of our current work.

We acknowledge that we have not explored different configuration parameters for evolutionary and non-evolutionary rule learning algorithms in this study. Therefore, the performance of some algorithms, reported in this paper, is likely to be sub-optimal. In future, we plan to explore configuration parameters of all algorithms to optimize performance.

We also plan to unify all datasets, used in this study, to create one multi-class dataset. This will make the dataset more challenging in terms of accuracy (due to multi-class problem), comprehensibility of rules (due to large rule sets) and processing overhead (due to increase in the total number of instances). Moreover, we plan to extend our comparative study on other real-world malware datasets as well.

Acknowledgements

The authors would like to thank the anonymous reviewers for providing valuable and insightful comments which helped us to be more clear about the experimental setup used in this study.

This work is supported by the National ICT R&D Fund, Ministry of Information Technology, Government of Pakistan. The information, data, comments, and views detailed herein may not necessarily reflect the endorsements of views of the National ICT R&D Fund.

6. REFERENCES

[1] J.H. Holland, L.B. Booker, M. Colombetti, M. Dorigo, D.E. Goldberg, S. Forrest, R.L. Riolo, R.E. Smith, P.L. Lanzi, W. Stolzmann, S.W. Wilson, "What Is a Learning Classifier System?", Internatinoal Workshop on Learning Classifier Systems (IWLCS), Volume 1813 of Lecture Notes in Artificial Intelligence, pp. 3-32, Springer, 2000.

[2] S.W. Wilson, "Classifier fitness based on accuracy", Evolutionary Computation, 3(2), pp. 149-175, MIT Press, 1995.

[3] E.B. Mansilla, J.M.G. Guiu, "Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks Ester", Evolutionary Computation, 11(3), pp. 209-238, MIT Press, 2006.

[4] J. Bacardit, J.M. Garrell, "Evolving Multiple Discretizations with Adaptive Intervals for a Pittsburgh Rule-Based Learning Classifier System", Genetic and Evolutionary Computation Conference (GECCO), Volume 2724 of Lecture Notes in Computer Science, pp. 1818-1831, Springer, USA, 2003.

[5] J. Bacardit, J.M. Garrell, "Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach Learning Classifier System", International Workshop on Learning Classifier Systems (IWLCS), Volume 4399 of Lecture Notes in Artificial Intelligence, pp. 59-79, Springer, UK, 2007.

[6] A. Gonzalez, R. Perez, "SLAVE: A genetic learning system based on an iterative approach", IEEE Transactions on Fuzzy Systems, 7(2), pp. 176-191, 1999.

[7] W.W. Cohen, "Fast Effective Rule Induction", 12th International Conference on Machine Learning (ICML), pp. 115-123, Morgan Kaufmann, USA, 1995.

[8] W.W. Cohen, Y. Singer, "A simple, fast, and effective rule learner", 16th National Conference on Artificial Intelligence, pp. 335-342, American Association for Artificial Intelligence (AAAI), USA, 1999.

[9] E. Frank, I.H. Witten, "Generating accurate rule sets without global optimization", 15th International Conference on Machine Learning (ICML), pp. 144-151, Morgan Kaufmann, USA, 1998.

Table 5: Overall comparison of all algorithms used in this study

Algorithm	Classification Accuracy	Size of Rule Set	Comprehensibility of Rule Set	Processing Overheads
XCS	Low	Large	Low	Medium
UCS	Medium	Large	Low	Small
GAssist-ADI	High	Small	High	Large
GAssist-Intervalar	Medium	Small	Medium	Large
SLAVE	High	Small	Medium	Large
RIPPER	**High**	**Small**	**High**	**Small**
SLIPPER	**High**	**Small**	**High**	**Small**
PART	**High**	**Small**	**High**	**Small**
C4.5 rules	**High**	**Small**	**High**	**Small**
RIONA	Medium	-	-	Large

[10] J.R. Quinlan, "MDL and Categorical Theories (Continued)", 12th International Conference on Machine Learning (ICML), pp. 464-470, Morgan Kaufmann, USA, 1995.

[11] G. Gora, A. Wojna, "RIONA: A New Classification System Combining Rule Induction and Instance-Based Learning", Fundamenta Informaticae, 51(4), pp. 369-390, IOS Press, 2002.

[12] A.O. Puig, J. Casillas, E.B. Mansilla, "Genetic-based machine learning systems are competitive for pattern recognition", Evolutioanry Intelligence, 1(3), pp. 209-232, Springer, 2008.

[13] J. Bacardit, M.V. Butz, "Data Mining in Learning Classifier Systems: Comparing XCS with GAssist", International Workshop on Learning Classifier Systems (IWLCS), Volume 4399 of Lecture Notes in Artificial Intelligence, pp. 282-290, Springer, UK, 2007.

[14] E. Bernado, X. Llora, J.M. Garrell, "XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks", Advances in Learning Classifier Systems, Volume 2321 of Lecture Notes in Computer Science, pp. 115-132, Springer, 2002.

[15] F. Kharbat, L. Bull, M. Odeh, "Mining Breast Cancer Data with XCS", Genetic and Evolutionary Computation Conference (GECCO), pp. 2066-2073, ACM Press, UK, 2007.

[16] A.O. Puig, E.B. Mansilla, "Evolutionary rule-based systems for imbalanced data sets", Soft Computing - A Fusion of Foundations, Methodologies and Applications, 13(3), pp. 213-225, Springer, 2008.

[17] K. Shafi, T. Kovacs, H.A. Abbass, W. Zhu, "Intrusion detection with evolutionary learning classifier systems", Natural Computing, Springer, 2007.

[18] K.C. Tan, Q. Yu, C.M. Heng, T.H. Lee, "Evolutionary computing for knowledge discovery in medical diagnosis", Artificial Intelligence in Medicine, 27(2), pp. 129-154, Elsevier, 2003.

[19] Y. Gao, J.Z. Huang, H. Rong, D.Q. Gu, "LCSE: Learning Classifier System Ensemble for Incremental Medical Instances", International Workshop on Learning Classifier Systems (IWLCS), Volume 4399 of Lecture Notes in Computer Science, pp. 93-103, Springer, UK, 2007.

[20] J. Alcala-Fdez, L. Sanchez, S. Garcia, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernandez, F. Herrera, "KEEL: a software tool to assess evolutionary algorithms for data mining problems", Soft Computing, Volume 13, pp. 307-318, Springer, 2009.

[21] C. Blake, E. Keogh, C. Merz, "UCI repository of machine learning databases", 1998, available at `www.ics.uci.edu/mlearn/MLRepository.html`.

[22] R. Rivest, "Learning Decision Trees", Machine Learning, Vol. 2, pp. 229-246, 1987.

[23] S.W. Wilson, "Compact rulesets from XCSI", International Workshop on Advances in Learning Classifier Systems, Volume 2321 of Lecture Notes in Artificial Intelligence, pp. 197-210, Springer, 2002.

[24] P.W. Dixon, D.W. Corne, M.J. Oates, "A ruleset reduction algorithm for the XCSI Learning Classifier System", Volume 2661 of Lecture Notes in Computer Science, pp. 20-29, Springer, 2004.

[25] C. Fu, L. Davis, "A modified classifier system compaction algorithm", Genetic and Evolutionary Computation Conference (GECCO), pp. 920-925, Morgan Kaufmann, USA, 2002.

[26] A.O. Puig, E.B. Mansilla, "Analysis of reduction algorithms for XCS classifier system", Recent Advances in Artificial Intelligence Research and Development, pp. 383-390, IOS Press, 2004.

[27] Microsoft Portable Executable and Common Object File Format Specification, Windows Hardware Developer Central, Updated March 2008.

[28] F-Secure Virus Description Database, available at `http://www.f-secure.com/v-descs/`.

[29] T.M. Cover, J.A. Thomas, "Elements of Information Theory", Wiley-Interscience, 1991.

[30] VX Heavens Virus Collection, VX Heavens website, available at `http://vx.netlux.org`.