A novel rule generator for intrusion detection based on frequent subgraph mining

Novedoso generador de reglas para la detección de intrusos basado en minería de subgrafo frecuentes

Vitali Herrera-Semenets[1]* Andres Gago-Alonso[1]

Recibido 28 de julio de 2015, aceptado 14 de julio de 2016
Received: July 28, 2015 Accepted: July 14, 2016

ABSTRACT

The current development of technologies has boosted the use of telecommunication services. This fact brings an increase in the volumes of data generated in telecommunication companies. Such data need to be processed in order to detect potential intruders or fraud. The rule evaluation techniques are widely used in these application contexts due to their high effectiveness over known attacks. The incorporation of an automatic rule generator allows it to obtain rules in large volumes of data, for assisting information analysts; thus, the accuracy of intrusion detection is increased. In this paper, an automatic rule generation method is presented, including a strategy based on processing the patterns extracted from a training set and building classification rules. Finally, our proposal is evaluated and compared regarding other classifiers, achieving good results.

Keywords: Automatic rule generation, frequent subgraph mining, intrusion detection.

RESUMEN

El desarrollo actual de las tecnologías ha impulsado el uso de servicios de telecomunicaciones. Este hecho implica un aumento en los volúmenes de datos generados en las empresas de telecomunicaciones. Dichos datos requieren ser procesados con el fin de detectar potenciales intrusos o fraudes. Las técnicas de evaluación regla son ampliamente utilizadas en estos contextos de aplicación por su alta efectividad sobre ataques conocidos. La incorporación de un generador automático de reglas en dichas técnicas, permite obtener reglas a partir de grandes volúmenes de datos, lo cual contribuye al trabajo de los analistas de información e incrementa la precisión durante la detección de intrusos. En este trabajo, se presenta un método de generación automática de reglas, que comprende una estrategia basada en el procesamiento de patrones extraídos de un conjunto de entrenamiento y la conformación de reglas de clasificación. Finalmente, nuestra propuesta es evaluada y comparada con respecto a otros clasificadores, alcanzando buenos resultados.

Palabras clave: Generación automática de reglas, minería de subgrafos frecuentes, detección de intrusos.

[1] Advanced Technologies Application Center (CENATAV). 7a # 21406. Playa, C.P. 12200. La Havana, Cuba.
E-mail: vherrera@cenatav.co.cu; agago@cenatav.co.cu
* Corresponding Author

INTRODUCTION

Technological advances in telecommunications boosts the creation of new services attracting more users, including malicious users known as intruders. Intruders execute unauthorized activities using telecommunications services, causing millions in losses and damaging the prestige of the affected company. The data to be analyzed in intrusion detection scenarios have particular characteristics; for example: high volume of instances, high number of features, systems under attack (this causes the intruders to modify their attacks in order to avoid detection), multiclass categorization problem and data are generated at high speeds.

There are several techniques for intrusion detection [2]. Some of the commonly used techniques are based on rule evaluation, automatic rule generation (ARG), anomalies detection, social networks analysis, Bayesian networks, neural networks, or hybrid systems. Such techniques can be based on supervised approach [3] (requires a supervised training collection) or unsupervised approach [4] (there is no a priori knowledge). Our proposal is based on the supervised approach.

Early intrusion detection reduces damages to the affected companies. Rule evaluation is one of the commonly used techniques, since it can be applied in real time, achieving high effectiveness in already known intrusion behaviors.

However, the techniques based on rule evaluation present some problems. One of them is that the analyst must manually define the specific rules for each class, which is an extremely complex and time consuming task [19]. Another problem is that they fail to detect slight changes in data [1], which could indicate the occurrence of criminal activity. In order to solve the above problems, several techniques based on ARG have been proposed.

In this paper, an automatic rule generation method for intrusion detection based on frequent subgraph mining is presented. Our approach consists of two algorithms: Rule Generation (RG) algorithm and Rule Filtering (RF) algorithm. Initially, the RG algorithm processes the frequent subgraphs extracted from the training set and generates rules from them. Then, the RF algorithm performs a filtering process by selecting the best rules. Finally, new instances are classified according to a set of filtered rules, using a novel evaluation strategy.

BACKGROUND

This section describes the basic concept of rule, as well as the related work. In addition, the method used to extract patterns using frequent subgraph mining is presented.

The rule definition

Let $F_1, F_2,\ldots F_k$ be sets whose elements are known as features. The integer k, $k > 1$, represents the number of features that characterize the data to be processed. Indeed, all rule discovery algorithms process sequences or stream of instances, where each instance $n = (f_1, f_2, f_k)$ is a vector belonging to the universe $U = F_1 \times F_2 \times \ldots \times F_k$, where $f_i \in F_i$ for every $1 \leq i \leq k$.

Let $\pi_1, \pi_2, \ldots, \pi_k$ be functions such that $\pi: U \rightarrow F_i$ so that given an instance $n = (f_1, f_2, \ldots, f_k) \in U$ to $\pi(n) = f_i$, for each $1 \leq i \leq k$. These functions in the literature are known as projection functions.

A condition in the universe U is a m: $U \rightarrow F_i$ $\{0, 1\}$, defined from three elements $\pi \oplus$ and f_i where π is a projection function, $\oplus$ is a comparison operator valid for f_i elements (if f_i is a numerical set $\oplus \in \{<, \leq, >, \geq, = \ldots\}$), and $f_i \in F_i$. An instance n satisfies the condition m when the comparison between $\pi(n)$ and f_i performed by $\oplus$ returns true, in that case $m(n) = 1$, otherwise $m(n) = 0$.

A formula in the universe U is also a function that takes values $\{0, 1\}$ which is defined recursively based on the following principles:

- If m is a condition in U then $\rho = m$ is a formula.
- If g is a formula in U then $\rho = (g)$ is another formula, so that $\rho(n) = g(n)$ for all $n \in U$.
- Given two formulas U, g_1, g_2, we have:
  - $\rho = g_1$ and g_2, is a formula, where $\rho(n) = g_1(n) \wedge g_2(n)$ for all $n \in U$.
  - $\rho = g_1 \mathrm{V}\, g_2$ is a formula where $\rho(n) = g_1(n) \vee g_2(n)$ for all $n \in U$.

A rule in the universe $U \times C$, is represented by $\left\langle \overleftarrow{r}, \overleftarrow{r} \right\rangle$, where C = $\{c_1, c_2, \ldots c_1\}$ is a set of classes with l being an integer $l \geq 2$, $\overleftarrow{r}$ a formula in U and $\overleftarrow{r}$ is an equality condition defined on the corresponding component classes (C). For simplicity, a rule is usually represented as $\left\langle \overleftarrow{r}, c_i \right\rangle$, where c_i. The intuitive meaning of a rule is that the fulfillment of the $\overleftarrow{r}$ premise in an instance implies that such instance belongs to the class c_i.

For example, let it assume that we have a rule r in the form: ***if*** ($f_1 \geq 0.2$ and $f_1 \leq 5.0$) and f_3 = *"http" then"attack"*; where $\overleftarrow{r}$ ($f_i \geq 0.2$ and $f_1 \leq 5.0$) and f_3 *"http",* as well as c_1 = *"attack"*. Considering $\overleftarrow{r}$ and c_1, an instance n = (0.3, "http", *"UDP"*) is labeled as *"attack"*, since $\overleftarrow{r}$ is fulfilled in n.

Related work

The ARG techniques [5] are useful for dealing with various challenges in many application contexts, especially for intrusion detection or fraud detection. These techniques provide a better understanding to analysts, since the outputs are defined as expressions in an understandable language [5], unlike other ones considered as black boxes. The ARG methods discussed in this paper can be clustered into four categories: greedy algorithms, conceptual clustering, technique to obtain fuzzy rules and decision trees.

The greedy algorithms (AQ21 [7], X2R [8], RIPPER [9], PART [6], AntMiner [10]) [5] perform an iterative process over the training collection. The first step consists in selecting a set of instances from the training collection to generate rules. After the rules are created, a filtering process is executed, and the filtered rules are added to a rule set. Subsequently, the selected instances above are removed from the training collection and the iterative process repeats until there are no more training instances. Finally, a filtering rule set is obtained.

In general, the rules, obtained from ARG techniques, generally define unambiguous boundaries for the features of the data to be analyzed. The unambiguous boundaries can often be fraudulent activities laying, but not exceeding, the threshold. The FURIA algorithm [11] presents an interesting proposal for dealing with this situation. This approach performs an adaptation of the RIPPER algorithm, allowing the computation of a first rule set. The obtained rules are transformed into fuzzy rules, by processing the conditions associated to numerical attributes.

There are several algorithms based on creating rules coming from decision trees, for example C4.5 [12], C5.0 [13] and CART [14]. A rule is generated for each leaf, following the path from the root to leaf. The first step of these algorithms is to generate a decision tree. The criteria used to construct the decision tree can vary depending on the proposal [5]. A pruning process is applied to the generated a decision tree, and a collection of trees is obtained. According to the used method in the previous step, the collection may comprise one or more trees. Afterwards, an optimization procedure is performed in order to select the optimal tree from the collection. Finally, a rule set from the selected tree is obtained.

The method presented in [15] combines conceptual clustering and natural induction to discover rules. In this method, the data associated to taxes are clustered by a conceptual algorithm. Then, the fraudulent instances of each cluster are used as training collection to learn new rules. The output contains simple rules describing regular and fraudulent instances.

A framework based on frequent subgraph mining

Recently, a framework for intrusion detection based on frequent subgraph mining was reported in literature [18]. During the training stage, this framework processes the training collection through two modules to get frequent subgraphs (see Figure 1).

The preprocessing module is the first to be executed. This step is performed to reduce the dimensionality of the training collection and relabel the values of each selected feature. The relabeling algorithm assigns a unique numerical label for each non numerical feature value. Furthermore, it creates a range of numerical values which are represented by a unique numerical label. These labels are assigned to each numerical feature value. Then, the redundant instances are removed. This module allows obtaining a smaller and quality data collection to improve the effectiveness in the patterns extraction process.

The reduced collection is then processed by the graph mining module. In this module, each instance is represented as a star graph [18]. In a graphical representation (see Figure 2), a star graph has a central vertex representing the instance itself and a vertex for each corresponding feature f_i connected

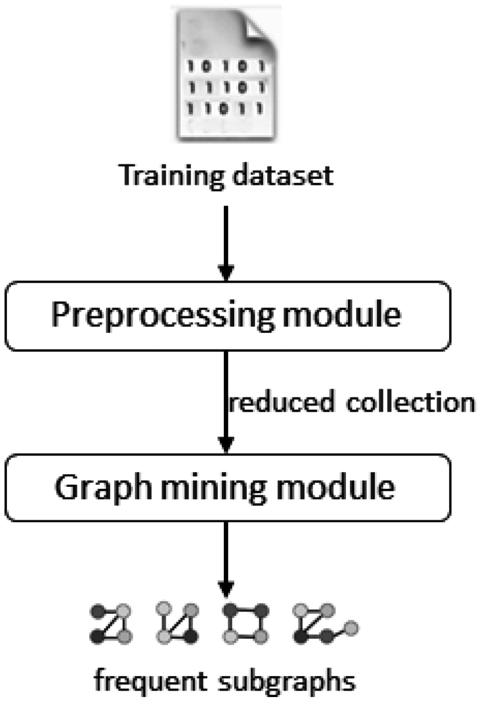

Figure 1. Training stage diagram used from the framework [18].

to the center. The edge label is the feature index *i*, and the vertex label is the value of the corresponding feature. Finally, frequent subgraphs (patterns) are obtained which will represent the pattern set *P*.

In this paper, we use the first two modules of the training stage in this framework as a basis to obtain a pattern set *P* for rule generation.

RULE GENERATOR

In this section, the proposed automatic rule generator and the rule evaluation strategy are introduced. The proposed method comprises of two algorithms: RG algorithm and RF algorithm. Both algorithms are discussed below as well the rule evaluation strategy.

RG algorithm

After obtaining a pattern set *P* using the framework described above, the RG algorithm is performed. This algorithm starts with an iterative process (see Figure 3, line 2). For each pattern *p* in pattern set *P*, the class *c* with higher occurrence of *p* is selected by the function "Search-Most-Covered-Class" (see Figure 3, line 3).

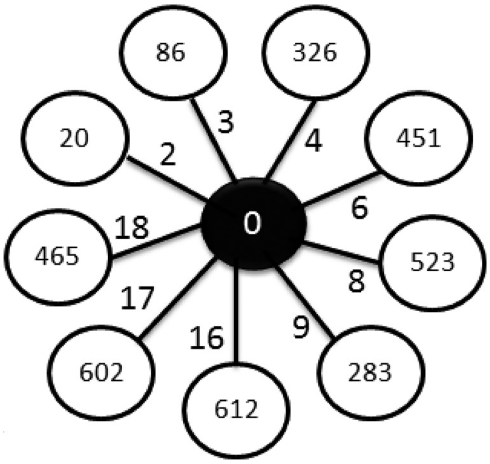

Figure 2. Star graph example.

Then, the percentage of occurrences c_p of *p* in *c* is calculated as $c_p = \frac{o_c}{o_t}$ in the function "Calculate-Percent-Covered-Class" (see Figure 3, line 4); where the variable o_c represents the number of occurrences of *p* in *c*, and the variable o_t represents the total number of occurrences of *p* in all classes.

RG Algorithm (P, L_1)

```
   Input: p - pattern set, L1- occurrence threshold
   Output: R- rule set
1  R← ∅;
2  foreach $p \in P$ do
3      $c_p$ ←Search-Most-Covered-Class($p$);
4      $c_p$ ← Calculate-Percent-Covered-Class (c);
5      if $c_p > L_1$ then
6          $r$ ← Create-Rule ($p$, $c$);
7          $R \leftarrow R \cup \{r\}$
8      end
9  end
10 return R;
```

Figure 3. Rule Generation algorithm.

If the percentage of occurrences c_p is greater than the occurrence threshold L_1, then a new rule *r* is created by the function "Create-Rule" according to the pattern *p* and the class *c* (see Figure 3, line 6). L_1 can take values within range [0, 1].

Here a pattern is represented as a rule. For example, for the pattern *p* shown in the Figure 4, assuming that the most covered class c was "DoS", the label "601" indicates the range [0.7, 1.0], and the label "271" represents the protocol "tcp", the generated rule r would be: *if* (≥ 0.7 *and* $f_3 \leq 1.0$) *and* f_4 = "tcp" then "*DoS*".

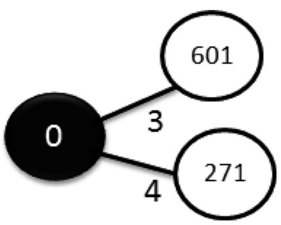

Figure 4. "DoS" pattern example.

The created rule r is added to the rule set R. Once all the iterations have finished, the rule set R containing all the generated rules is returned.

RF algorithm

The obtained rule set R is processed by the RF algorithm (see Figure 5). This algorithm requires the entire training collection D_t from where the pattern set P was obtained. Furthermore, it takes a precision threshold L_2 defined by an analyst, which can take values within range [0, 1]. We use the entire training collection, because the used framework to generate the patterns set P, reduces the training collection. Therefore, the fact of using the entire training collection provides more information to compute the precision value *r precision* of the rule r.

This algorithm starts with an iterative process (see Figure 5, lines 1-6). For each instance d, where $d \in D_t$, each rule r is evaluated, where $r \in R$ (see Figure 5, lines 2-5). In the Calculate-Precision function (see Figure 5, line 3), if r covers d, then *r.precision* value is calculated as $r.precision = \frac{r.correct}{r.total}$.

The variable *r.precision* represents the number of instances correctly covered by r, and the variable *r.total* represents the total number of covered instances by r. After calculate *r.procision* value, the rule r with its new precision value and the new number of instances correctly covered is updated in the rule set R.

Once all the instances in D_t are processed, another iterative process starts in order to select the most

```
RF Algorithm ($R, D_t, L_2$)
Input: $R$- rule set, $D_t$- training collection, $L_2$- precision threshold
Output: $R$- rule set
1  foreach $d \in D_t$ do
2      foreach $r \in R$ do
3          $r$. correct, $r$. presicion ⟸ Calculate- presicion threshold
           Precision($r$, $d$);
4          $R \leftarrow R\ U\ \{r\}$
5      end
6  end
7  $R_f \leftarrow \varnothing$;
8  foreach $r \in R$ do
9      if $r.precision > L_2$ then
10         $r_1 \leftarrow$ Exist ($r$, $Rf$);
11         if $\in$ then
12             if $r.precision > r_1.presicion$ then
13                 $R_f \leftarrow R_f \setminus \{r_1\}$
14                 $R_f \leftarrow R_f\ U\{r\}$
15             end
16         end
17         else
18             $R_f \leftarrow R_f\ U\{r\}$
19         end
20     end
21 end
22 $R_f \leftarrow$ Generalize-Rules ($R_f$);
23 $R_f \leftarrow$ Sort ($R_f$);
24 return $R$;
```

Figure 5. Rule Filtering algorithm.

representative rules (see Figure 5, lines 8-21). For each rule r, its precision value *r.precision* is compared against the precision threshold L_2. If *r.precision* exceeds L_2, then the function Exist (see Figure 5, line 10) search in the filtered rule set R_f, a rule r_1 (where $r_1 \in R_f$) that its premise $\left(\overline{r_1}\right)$ is the same as that of $\left(\overline{r_1}\right)$. If there is not a rule r_1, then r is added to R_1. But, if there exist a rule r_1, then a comparison is performed between precision values. In case of *r.procision* value is greater than r_1*.procision*, then r_1 is removed from R_f, and r is added to R_f.

When the iterative process concludes, the next step is to generalize rules in R_f using the function Generalize-Rules (see Figure 5, line 22). This function searches for rules that share at least one same conditions and define the same class. When it finds two rules with these characteristics a merger takes place.

Such merger is performed by keeping the identical conditions and logically adding those different. For example, assume we have two rules: the generated rule r from the presented pattern in Figure 4, and a rule r_1: ***if*** ($f_3 \geq 0.7$ ***and*** $f_3 \leq 1.0$) ***and*** f_6 = *"smtp"* ***them*** *"DoS"*; the result of the merger between r and r_1 is: ***if*** ($f_3 \geq 0.7$ ***and*** $f_3 \leq 1.0$) and (f_4 = *"tcp"* ***or*** f_6 = "smtp") *then "DoS"*. This procedure not only allows us to obtain more general rules, but also reduce the final set of rules, providing more effectiveness in the rules evaluation process.

Using the Sort function (see Figure 5, line 23), the obtained rule set is sorted according to the precision of the rules. Thus, the rules with highest precision value will be the first in order. If there are two rules with the same precision value, then the one with greater number of instances correctly covered during the filtering process will go first in order. Finally, a filtered rule set R_f with the most representative rules is returned.

Rule evaluation

Like others rule-based techniques [19], our method is supplemented with a rule evaluation (RE) strategy. The goal of this strategy is to assign a label to a new instance. The RE algorithm starts with an iterative process (see Figure 6, lines 3-10). Each rule r, where $r \in R_f$, is evaluated over a new instance d, using Evaluate-Rule function (see Figure 6, line 4).

This function returns true if r covers d. When the variable *cover* take the value *true*, then the iterative process stops and the label l takes the value of the decision part of the rule $\overleftarrow{r}$. If no rule in R covers d, then the algorithm does not assign a specific class, and l takes the value "abstain". Finally, the label l that defines the new instance d is returned.

```
RE Algorithm ($R_f$, $d$)
     Input: $R_f$ - filtered rule set, $d$ - new instances
     Output: $l \leftarrow$ "label"
1    $l \leftarrow$ "abstain"
2    cover $\leftarrow$ false;
3    foreach $r \in R_f$ do
4        cover $\leftarrow$ Evaluate-Rule ($r$, $d$);
5        $R \leftarrow R \cup \{r\}$;
6        if cover = true then
7            $l \leftarrow \overleftarrow{r}$;
8            break
9        end
10   end
11   return $l$;
```

Figure 6. Rule Evaluation algorithm.

EXPERIMENT

In this section, the dataset used for evaluating the proposed method is analyzed and the experimental results achieved are presented.

KDD Cup '99 dataset

In our experiment we use the KDD '99 Dataset [16], which provides connection records (instances) generated by a simulation of a military network. This dataset contains two labeled collections: training with 4, 898, 431 instances and testing with 311,029 instances. Each instance contains 41 features, which can be continuous or discrete, and is labeled as either normal or an attack, with exactly one specific type of attack.

The simulated attacks are clustered into the following four categories: surveillance and other probing (Probe), denial of service (DoS), unauthorized access to local root privileges (U2R) and unauthorized access from a remote machine (R2L). Therefore, unlike other datasets which have only one specific attack type (e.g. CAIDA "DDoS Attack 2007" dataset [17]), the KDD '99 dataset contains instances of four different attacks types.

Experimental results

Experiments were performed on a machine with a Quad Core processor at 2.5 GHz and 4 GB of RAM. The algorithms are implemented using Python and ANSI-C programming languages. To run the RG algorithm is required a pattern set P and a defined occurrence threshold L_1. The pattern set P is obtained by the framework based on frequent subgraph mining above. In order to obtain a substantial number of rules to represent all existing classes, we seek a small value for occurrence threshold (in our case $L_1 = 0.3$). Once the input parameters are ready, we proceed to generate the rule set R. The total number of rules generated was $|R| = 3081$.

The obtained rule set R is processed by the RF algorithm. This step is based on preserving the rules that best describe the existing classes; to do this, a value close to 1 must be defined for the precision threshold (in our case $L_2 = 0.7$). To perform the RF algorithm, the same training collection D_t used by the framework to extract patterns most be provided. Then, the rule set R is significantly reduced to a filtered rule set R_f, where $|R_f| = 41$.

With the filtered rule set, we proceed to evaluate the rules on the testing collection using RE algorithm. In order to compare our results, we process the testing collection with different classifiers using the reduced training collection (see Figure 1) to build the classification models. Some of the methods discussed in Related Work Section could not be included in the comparison because they are not publicly available.

In our experiments, we evaluate the performance of classifiers such as C5.0 and others from Weka platform [20], which are: decision table/naive bayes hybrid classifier (DTNB), Id3, JRip (RIPPER), nearest-neighbor-like algorithm using non-nested generalized exemplars (NNge), PART, PRISM, Ripple-Down Rule learner (RIDOR) AND SimpleCart (CART). The achieved results over the same KDD '99 test collection are shown in the Table 1 (our approach is represented as RE). Moreover, the reported results in [18] using J48graft, classification via Regression and SMO classifiers are included in the Table 1. The accuracy achieved for an attack category is computed as is shown in equation (1).

$$accuracy(category_i) = \frac{true\_positive}{total\_category\_instances} \times 100 \quad (1)$$

The variable $category_i$ indicates the i-th attack category to be calculated it accuracy, *true_ positive* represent the number of instances from $category_i$ correctly classified during the test, and *total_category_instances* is the total number of instances labeled as $category_i$ in the test collection. The accuracy for "normal" category is computed using equation (2).

Table 1. Accuracy achieved using different classifiers.

Classifier	Normal (%)	Probe (%)	DoS (%)	U2R (%)	R2L (%)
C5.0	99.88	46.45	79.14	0	0
CART	99.99	45.6	71.6	0	0
DTNB	99.8	65.7	7.9	3.5	0
Id3	99.91	18.2	65.8	3.5	0
J48graft [18]	99.9	43.73	71.41	0	0
NNge	4.6	38.1	90.4	23.7	1.5
PART	99.95	41	79.1	0	0
PRISM	99.93	18.3	65.7	3.1	0
RE	72.29	55.14	97.34	0	0
Regression [18]	55.09	51.44	71.44	4.39	27.7
RIDOR	99	58.3	71.9	7.5	0
RIPPER	46.8	68.7	79.8	0.9	0
SMO [18]	54.97	53.29	71.6	3.95	28.2

$$accuracy(normal) = \frac{true\_negative}{total\_normal\_instances} \times 100 \quad (2)$$

The variable *true_negatine* represent the number of instances from "normal" category correctly classified during the test, and *total_normal_instances* is the total number of instances labeled as "normal" in the test collection.

From the obtained results, it can be said, that none of the filtered rules R_f are representative of U2R and R2L attack categories. This is because, the precision value of such attacks representative rules is below the precision threshold L_2. Moreover, our approach outperforms other classifiers in terms of accuracy in DoS attack category.

Since our approach is able to abstain, misclassifications tend to decrease (see Table 2). In Table 3, can be seen that the misclassification rate in our proposal is considerably lower than those reported by other classifiers. Another important aspect to note is the overall accuracy, given by equation (3).

$$overall\_accuracy = \frac{(total\_tp + true\_negative)}{total\_test\_instances} \times 100 \quad (3)$$

In such equation, the variable *total_tp* represents the number of attack instances correctly classified during the test, and *total_test_instances* indicates the number of instances in the test collection. The

Table 2. Summary of the achieved results using our approach.

	Number of instances
Correctly classified	269.837
Abstentions	36.926
Misclassifications	4.266

Table 3. Overall accuracy and misclassification rate achieved using different classifiers.

Classifier	Overall accuracy (%)	Misclassification rate (%)
C5.0	78,56	21,44
CART	73,01	26,99
DTNB	26,19	73,81
Id3	68,32	31,66
J48graft [18]	72,82	27,18
NNge	68,3	31,7
PART	78,49	21,51
PRISM	68,23	31,76
RE	86,76	1,37
Regression [18]	65,66	34,34
RIDOR	73,21	26,79
RIPPER	68,98	31,02
SMO [18]	65,56	34,44

achieved results shown in the Table 3, prove that our proposal with 86.76 % of overall accuracy and 1.37 % of misclassification rate in general terms outperforms other analyzed classifiers. Inclusive, if abstentions are considered as misclassification, even so, the misclassification rate with 13.24% would be lower than those reported by the analyzed classifiers.

The results achieved by our approach show that it can be applied in scenarios that require minimal misclassifications. Since it is a rule-based approach [19], it can be incorporated as a module of a security system to identify online criminal activities in data streams. Also, the instances labeled as “abstain”, can be used to generate patterns. Then, from these patterns new rules can be generated that define new classes, or existing classes that have changed their concept.

CONCLUSIONS

In this paper, a new method for generating rules from frequent subgraphs was presented. This proposal also includes an algorithm for filtering rules, allowing it to merge the generated rules, building generalized rules and reducing the resulting output. Our rule evaluation process included a strategy to abstain, which reduces the number of misclassification.

The experimental results show the effectiveness of our proposal for detecting intruders, improving the results obtained by other classifiers in terms of overall accuracy and misclassification rate. This represents an indicator to consider, for its possible application as part of an intrusion detection system.

As future work, we intend to extend our proposal to a dynamic method. This means that the method may be able to generate new sets of rules taking into account the previously generated rules.

REFERENCES

[1] H. Fanaee-T and J. Gama. “Event labeling combining ensemble detectors and background knowledge”. Progress in Artificial Intelligence. Springer, pp. 1-15. 2013.

[2] V. Herrera-Semenets, M.A. Prado-Romero and A. Gago-Alonso. “Análisis de los métodos de detección de fraude en servicios de telecomunicaciones”. Technical Report RT 023. Serie Gris. Advanced Technologies Application Center (CENATAV). La Habana, Cuba. 2014.

[3] S.B. Kotsiantis. “Supervised Machine Learning: A Review of Classification Techniques”. In Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies. Amsterdam, The Netherlands, pp. 3-24. 2007.

[4] Z. Ghahramani. “Unsupervised learning”. Advanced Lectures on Machine Learning. Springer, pp. 72-112. 2004.

[5] V. Herrera-Semenets and A. Gago-Alonso. “Búsqueda automática de reglas para detección de fraudes en flujos de eventos”. Technical Report RT 29. Serie Gris. Advanced Technologies Application Center (CENATAV). La Habana, Cuba. 2015.

[6] E. Frank and I.H. Witten. “Generating accurate rule sets without global optimization”.

University of Waikato, Department of Computer Science. 1998.

[7] S.M. Ryszard, A.K. Kenneth, P. Jaroslaw, W. Janusz, M. Scott and S. Doug. "Natural induction and conceptual clustering: A review of applications". Reports of the Machine Learning and Inference Laboratory, George Mason University. Fairfax, VA, USA. 2006.

[8] H. Liu and S.T. Tan. "X2R: A fast rule generator". In Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada, 1995.

[9] W.W. Cohen. "Fast effective rule induction". In Proceedings of the 12th International Conference on Machine Learning, Tahoe City, pp. 115-123. 1995.

[10] R.S. Parpinelli, H.S. Lopes and A.A. Freitas. "An ant colony algorithm for classification rule discovery". Data Mining: A Heuristic Approach. Vol. 208, pp. 191-208. 2002.

[11] J. Hühn and E. Hüllermeier. "FURIA: an algorithm for unordered fuzzy rule induction". Data Mining and Knowledge Discovery. Springer. Vol. 19, pp. 293-319. 2009.

[12] J.R. Quinlan. "C4.5: programs for machine learning". San Francisco, California, Morgan Kaufmann, pp. 302. 1993.

[13] M. Kuhn and K. Johnson. "Applied predictive modeling". Springer. New York, USA. 2013.

[14] R.L Lawrence and A. Wright. "Rule-based classification systems using classification and regression tree (CART) analysis". Photogrammetric engineering and remote sensing. Vol. 67 Nº 10, pp. 1137-1142. 2001.

[15] R.S. Michalski, K.A. Kaufman, J. Pietrzykowski, J. Wojtusiak, S. Mitchell and D. Seeman. "Natural Induction and Conceptual Clustering: A Review of Applications". Reports of the Machine Learning and Inference Laboratory. Vol. 1051. George Mason University. Fairfax, VA, USA. 2006.

[16] KDD Cup 1999. Computer network intrusion detection. [Online]. [cited April 3, 2015]. URL: http://sigkdd.org/kdd-cup-1999-computer-network-intrusiondetection

[17] The CAIDA UCSD. "DDoS Attack 2007" Dataset. [Online]. [cited April 3, 2015]. URL: http://www.caida.org/data/passive/ddos-20070804_dataset.xml

[18] V. Herrera-Semenets, N. Acosta-Mendoza, and A. Gago-Alonso. "A Framework for Intrusion Detection based on Frequent Subgraph Mining". In proceedings of The Second SDM Workshop on Mining Networks and Graphs: A Big Data Analytic Challenge (SDM-Networks 2015). In conjunction with 2015 SIAM international Conference on Data Mining (SDM15). Vancouver, BC, Canada. ISSN: 2167-0099. 2015.

[19] M. Deckert. "Incremental rule-based learners for handling concept drift: an overview". Foundations of Computing and Decision Sciences. Vol. 38 Nº 1, pp. 35-65. 2013.

[20] The University of Waikato. "Weka 3: Data Mining Software in Java". [Online]. [cited April 5, 2015]. URL: http://cs.waikato.ac.nz/ml/weka