A fast instance reduction algorithm for intrusion detection scenarios

Herrera-Semenets, Vitali; Hernández-León, Raudel; van den Berg, Jan

DOI
10.1016/j.compeleceng.2022.107963

Publication date
2022

Document Version
Final published version

Published in
Computers and Electrical Engineering

Computers and Electrical Engineering 101 (2022) 107963

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

A fast instance reduction algorithm for intrusion detection scenarios☆

Vitali Herrera-Semenets [a,*], Raudel Hernández-León [a], Jan van den Berg [b]

[a] Advanced Technologies Application Center (CENATAV), 7a ♯ 21406, Playa, C.P. 12200, Havana, Cuba
[b] Intelligent Systems Department, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

ARTICLE INFO

Keywords:
Instance selection
Data reduction
Intrusion detection
Data preprocessing
Data mining
Supervised classification

ABSTRACT

We live in a world that is being driven by data. This leads to challenges of extracting and analyzing knowledge from large volumes of data. An example of such a challenge is intrusion detection. Intrusion detection data sets are characterized by huge volumes, which affects the learning of the classifier. So there is a need to reduce the size of the training sets. Fortunately, inspection and analysis of available intrusion detection data sets showed that many instances are very similar and do not provide relevant information to the classification process. This prompted to look for possibilities to use a fast algorithm that, as much as possible, removes similar instances in intrusion detection data sets while enforcing the detection rate. In this work, a new fast instance reduction algorithm is presented. The proposed algorithm provides greater efficiency during the training stage, without significantly affecting the efficacy during the intrusion detection task.

1. Introduction

Intrusions in telecommunications networks are among the most used and harmful malicious activities. Statistical intelligence reports, provided by the Kaspersky Lab company, indicate that Distributed Denial of Service (DDoS) attacks[1] during the first quarter of 2020 include attacks that lasted up to twenty days affecting the availability of necessary services [1]. In this sense, the entities providing ICT[2]-based services require automated techniques to detect intrusions in the shortest possible time [2].

Intrusion detection in telecommunications networks, from a classification point of view, generally includes two steps: (1) building a classification model from a training data set and (2) using that model for classification, in this case, to detect intrusions. Intrusion detection data sets are characterized by having large volumes [3], which can affect both steps, since some classifiers may need to process the data several times, either during the training or the classification stage.

A straightforward way to reduce the time cost in the above two steps is to minimize the size of the training data set by using an instance selection algorithm. However, developing a fast and scalable solution that allows reducing the training set to efficiently build a classification model, providing greater efficiency to the intrusion detection process, without significantly affecting its efficacy, turns out to be a challenge.

To address this problem, some data sets collected in intrusion detection scenarios were analyzed, and the following annotations were performed:

☆ This paper is for regular issues of CAEE. Reviews were processed by Associate Editor Dr. S. Smys and recommended for publication.

* Corresponding author.
E-mail addresses: vherrera@cenatav.co.cu (V. Herrera-Semenets), rhernandez@cenatav.co.cu (R. Hernández-León), j.vandenberg@tudelft.nl (J. van den Berg).

[1] It is an attack that directs a large flow of information from different systems to a single target, forcing the denial of service of the target.

[2] Information and Communication Technologies.

https://doi.org/10.1016/j.compeleceng.2022.107963
Received 6 December 2021; Received in revised form 23 March 2022; Accepted 24 March 2022
Available online 21 April 2022

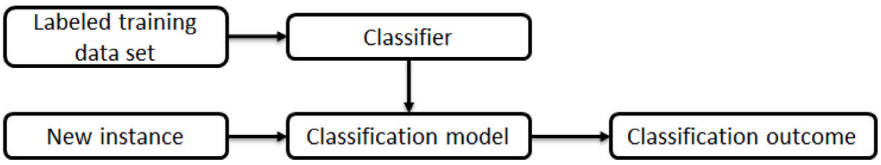

Fig. 1. Supervised approach scheme.

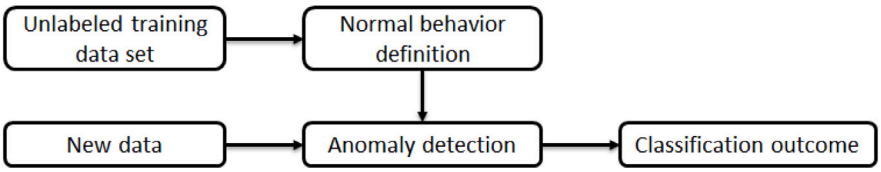

Fig. 2. Unsupervised approach scheme.

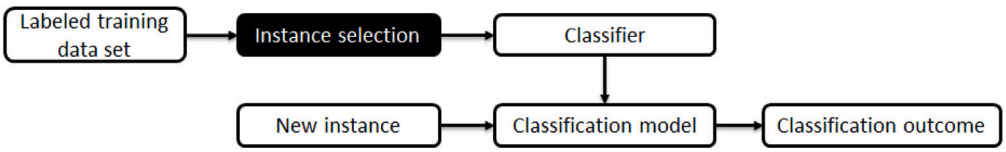

Fig. 3. Instance selection in the supervised scheme.

- Some instances of the training set are similar to each other, and differ only in few features with slightly different values, therefore, they do not provide additional information for building the classification model.
- Training data sets are composed of a large number of instances, which could number in the hundreds of thousands and even more than a million instances. Calculating the degree of similarity or some equivalent metric between each pair of instances requires a high time cost. Furthermore, many algorithms use up available resources, mainly RAM memory, when processing data sets with these characteristics [4].

Based on the previous observations, this work presents an algorithm for selecting instances that allows to reduce data sets from intrusion detection scenarios, with high efficiency and without significantly affecting the efficacy during the classification stage. The proposed algorithm uses a combination of heuristics that include a label generation stage and a relabeling stage, which allow to identify similar instances efficiently. Subsequently, similar instances are eliminated keeping only one representative. These stages provide the proposed instance selection algorithm with high performance for processing large volumes of data where other algorithms, of the same type, usually have poor performance.

The remainder of this paper is organized as follows. Related works are described in Section 2. The proposed algorithm is introduced in Section 3. In Section 4, the experimental results using different intrusion detection data sets are discussed. Finally, our conclusions and future work are outlined in Section 5.

2. Related work

A wide number of classification algorithms have been proposed for intrusion detection [5]. These techniques are based on two specific approaches: supervised and unsupervised.

The supervised approach requires prior domain knowledge [6], *i.e.*, a previously labeled data set is needed from which it is possible to extract representative patterns of malicious activities (build a classification model) to classify new instances (see Fig. 1).

On the other hand, the unsupervised approach usually detects abnormal actions from established normal behavior [7]. Normal behavior is modeled using an unlabeled training data set that reflects historical information (see Fig. 2). Once the normal behavior has been defined, it can be compared to the current behavior. This allows to determine if there have been any significant changes in the behavior that suggest an anomaly.

This work is focused on improving the performance of supervised classification algorithms by reducing the number of instances in the training data set. As shown in Fig. 3, the instance selection process is performed using the original data set and the output of this process (*i.e.*, the reduced data set) is used by the classifier to build the classification model.

The volume and quality of the data set are two important aspects to guarantee a good performance in a classifier. However, the time needed by a classifier to process a data set, composed of a large number of instances, can be affected, either at the training or classification stage [8]. This problem is most noticeable when using instance-based classifiers.

In general, not all instances in the training set provide relevant information to the classification process. In other words, the data set may contain unnecessary instances. There are two common types of unnecessary instances:

- Noisy instances: These are instances that can cause misclassifications of new instances. One of the main causes of this type of instance is the errors produced during the data collection process.
- Similar instances: These are instances whose descriptive features can be generalized by some other instances in the training set, which makes them unnecessary. These types of instances are very common in intrusion detection scenarios, although they also occur in data sets where instances of the same class are very similar.

Similar instances are usually caused by large volumes of data generated by telecommunications services. Suppose that, for an intrusion detection task, it is necessary to construct a classifier which allows to distinguish between attacks and normal instances. If the training data set is composed of instances associated with a Distributed Denial of Service attack, there must be a large number of instances that are very similar to each other, for example those originating from the same source [9]. These similar instances provide less information than other instances, that are generated by a different sources, but are also related to the DDoS attack, making them unnecessary for classifier training.

Suppose that a classifier is trained with n instances from a data set of size $m > n$, and that it can achieve the same efficacy as a classifier trained on all m instances. Then it would be more advantageous to train with the smaller data set, since it would provide greater spatial and time efficiency to the classifier.

Assuming the circumstances described above, there is a need to select only those instances that preserve or degrade the quality of the training set to a lesser extent.

The instance selection algorithms can be classified, according to their selection strategy, *Filter* [10] and *Wrapper* [11]. The *Filter* algorithms select the subset of instances independently from some classifier. On the other hand, the *Wrapper* algorithms select the subset of instances based on the results achieved by some classifier. In this approach, those instances that do not contribute to the efficacy of the classifier are not included in the reduced data set.

The non-use of classifiers within the selection process makes the *Filter* algorithms more efficient than the *Wrapper* ones. This means that the *Filter* algorithms perform better, in terms of execution time, than the *Wrapper* algorithms, since their selection criteria are not based on the classifier. However, this characteristic means that the *Wrapper* algorithms obtain greater efficacy, in terms of accuracy, than the *Filter* algorithms in classifiers for which they are designed. However, since the *Filter* algorithms are more efficient than *Wrapper* ones, their application is feasible in real intrusion detection scenarios, where large volumes of data need to be processed in the shortest possible time. Therefore, this research focuses on the *Filter* algorithms proposed to select instances in intrusion detection scenarios.

The improved intrusion detection approach described in [12] consists of an algorithm based on the correlation coefficient obtained between two instances. Considering the degree of statistical correlation between two variables, it is possible to obtain values in the range $[-1, 1]$. The closer to 0 the value, the lower the relationship between the evaluated instances. If the value obtained is close to -1 or 1, it indicates a negative or positive correlation respectively. A positive correlation is when the relationship between two variables is linear and direct, so that a change in one of them predicts the change in the other. This type of correlation is directly proportional. In the case of negative correlation (inversely proportional), the relationship between one variable and another is opposite, that is, when one variable changes, the other changes towards the opposite. Therefore, when one variable has high values, the other has low values, the more this value tends to -1, the more evident this covariation will be. This type of correlation is inversely proportional. The authors suggest that if the correlation value obtained between two instances exceeds a predefined threshold, these instances are largely statistically similar to each other, which is why it is possible to discard one of them. In this sense, given two instances $n_1 = (\check{a}_{1,n_1}, \check{a}_{2,n_1}, \ldots, \check{a}_{l,n_1})$ and $n_2 = (\check{a}_{1,n_2}, \check{a}_{2,n_2}, \ldots, \check{a}_{l,n_2})$, its correlation coefficient $r(n_1, n_2)$ is computed using Eq. (1), where the variable $\check{a}_{i,n_1}$ represents the value of the ith feature of instance n_1, while $\bar{a}_{n_1}$ is the average of the feature a values in n_1.

$$r(n_1, n_2) = \frac{\sum_{i=1}^{l}(\check{a}_{i,n_1} - \bar{a}_{n_1})(\check{a}_{i,n_2} - \bar{a}_{n_2})}{\sqrt{(\check{a}_{i,n_1} - \bar{a}_{n_1})^2}\sqrt{(\check{a}_{i,n_2} - \bar{a}_{n_2})^2}}. \tag{1}$$

A drawback of the [12] proposal relates to the fact that to calculate the correlation between two instances n_1 and n_2, the nominal features of n_1 must be equal to their corresponding ones at n_2. This implies that two instances having all their values equal except a nominal value, may be statistically similar, while it is not possible to calculate its correlation coefficient to decide to get rid of one, or not.

Some instance selection algorithms rely on clustering. Their objective is to transform the training data set D into clusters and then, to select the instances that define the centers of these clusters. The proposal introduced in [13] consists of an incremental clustering-based algorithm. The first step is to partition D into subsets $\check{D}_{c_j}$, where each one will be composed by instances associated with a single class $c_j \in C$, being C the set of classes present in D.

The next step is to calculate the centroid ζ_{c_j} of each subset $\check{D}_{c_j}$ and estimating the Euclidean distance $\varepsilon(\zeta_{c_j}, n_i)$, such that $n_i \in \check{D}_{c_j}$, between each centroid and its corresponding instances. The authors propose partitioning each subset again, but this time using a different criteria. For a subset $\check{D}_{c_1}$ belonging to a class c_1, the average $\bar{\varepsilon}$ of the distances calculated for each instance $n_i \in \check{D}_{c_1}$ is estimated concerning to its centroid ζ_{c_1}. Using the value of $\bar{\varepsilon}$, divide $\check{D}_{c_1}$ into two subsets, one consisting of instances that meet $\varepsilon(\zeta_{c_1}, n_i) > \bar{\varepsilon}$ and another composed of instances that meet $\varepsilon(\zeta_{c_1}, n_i) < \bar{\varepsilon}$. This process is repeated for each subset until a predetermined number of subsets per class is obtained. The next step is to select a representative instance from each subset. To estimate the degree of representativeness $\rho(n_i, \check{D}_{c_j})$ of an instance n_i in a subset $\check{D}_{c_j}$, the distance for each centroid of a class other than n_i is computed,

as well as the distance from the K-closest neighbors of n_i, which are added to the set $\check{D}_{n_i}$ (see Eq. (2)). The most representative instance of each cluster is included in the reduced data set $\check{D}$.

$$\rho(n_i, \check{D}_{c_j}) = \begin{cases} \sum_{l=1,\check{D}_{c_j} \neq \check{D}_{c_l}}^{|C|} \varepsilon(\zeta_{c_l}, n_i) & Si \sum_{l=1,n_l \in \check{D}_{n_i}}^{|\check{D}_{n_i}|} \varepsilon(n_l, n_i) = 0 \\ \dfrac{\sum_{l=1,\check{D}_{c_j} \neq \check{D}_{c_l}}^{|C|} \varepsilon(\zeta_{c_l}, n_i)}{\sum_{l=1,n_l \in \check{D}_{n_i}}^{|\check{D}_{n_i}|} \varepsilon(n_l, n_i)} & otherwise. \end{cases} \tag{2}$$

The algorithm presented in [14] also uses the Euclidean distance, but this time for calculating the closest K-neighbors of each instance $n \in D$. After this step, for each instance n, its effectiveness E_n is calculated using Eq. (3). The variable $\check{E}_n$ (reward) indicates the number of closest neighbors that belongs to the same class to n, and $\hat{E}_n$ (penalty) represents the number of closest neighbors with a different class to n.

$$E_n = \check{E}_n - \hat{E}_n. \tag{3}$$

Let μ be a predefined threshold. Those instances with $E_n \geq \mu$ are added to D_{SAFE}; otherwise they are added to a D_{TEMP}. Instances with $E_n < 0$ probably introduce some noise, and therefore, they can be eliminated from both sets D_{SAFE} and D_{TEMP}. If $D_{TEMP} \neq \emptyset$ then the process is repeated for each instance $n \in D_{TEMP}$. Once $D_{TEMP} = \emptyset$, then D_{SAFE} is partitioned into $\mid C \mid$, with C being the set of possible classes in D_{SAFE}. The RNNR [15] algorithm is applied to each of the obtained partitions. For an instance n, the RNNR algorithm searches all instances for which n is its closest neighbor. In this way, instances that can represent other instances of the same class are selected. Finally, all the selected instances of each partition are combined and the reduced set $\check{D}$ is obtained.

Another instance selection algorithm based on clustering approach is proposed in [10]. This algorithm consists of dividing the data set according to the number of classes. Then, the instances with the same class are grouped into several clusters following an established criteria [10]. From each cluster ς_i obtained, a defined quantity m of representative instances is selected. The process of selecting a representative instance consists of three fundamental steps. The first step is to calculate the density of each instance $n_j \in \varsigma_i$. The density of an instance is given by the number of neighbors it has with a distance less than a threshold. Then an instance is randomly selected from ς_i and added to the set of representative instances $\check{D}$. In the second step, for each instance $n_j \in \varsigma_i$, the smallest distance between n_j and the instances selected in $\check{D}$ is obtained. Finally, with the density and the minimum distance obtained, for each instance a probability is estimated. The rest of representative instances are selected based on a weighted probability distribution [10].

The *Filter* algorithms discussed above rely primarily on distance measures [10,13,14] and correlation measure [12] to perform instance selection. However, a distinctive aspect of the analyzed algorithms is that all of them look for similarities between the instances that compose the original training data set, and then select those instances that compose the reduced data set. Considering the volume of data handled in intrusion detection scenarios, computing the similarity or some equivalent metric for each pair of instances in a data set is a very expensive and inefficient process. For this reason, instance selection algorithms tend to have a high computational cost when applied to intrusion detection tasks, since their computational complexity is directly associated with the size of the training set. This characteristic may lead to the fact that the use of these algorithms is not feasible in practice. The process of reducing the data set can also result into quality improvement of the data, namely in cases where the removal of instances that do not provide relevant information to the classifier.

Instance selection algorithms mostly present solutions that involve a complex analysis of the data to finally select a subset of instances that represent the original data set. If the aforementioned is taken into account, the presence of a large number of similar instances describing the same class is a typical problem in intrusion detection scenarios. Therefore, our goal is to deal with this issue.

3. Proposal

The Fast Instance Reduction Algorithm (FIRA) proposed in this work consists of three fundamental stages: (1) label generation, (2) relabeling and (3) duplicated removing (see lines 1–3 of Algorithm 1).

Algorithm 1: FIRA(D, k)

Entrada: D: *Training data set*, k: *number of subsets*
Salida: $\check{D}$: *Reduced data set*

1 $L \leftarrow Label\_Generator(D, k)$ // *Labels are generated*
2 $D_L \leftarrow Relabeling(D, L)$ // *Feature values are relabelled*
3 $\check{D} \leftarrow Duplicated\_Removing(D_L)$ // *Duplicated instances are removed*
4 **return** $\check{D}$

The main step of the label generation stage is the discretization of the continuous features. The discretization process tries to go from an uncountably infinite set $\check{a}$ (see Eq. (4)) to a countable finite set ä (see Eq. (5)).

$$\check{a} = \{x \mid x \in [0, 1]\}, \ \mid \check{a} \mid = \infty, \tag{4}$$

$$\ddot{a} = \{x_j \mid x_j = j \triangle x\}, \mid \ddot{a} \mid = l, \tag{5}$$

where $\mid \breve{a} \mid$ represents the cardinality of the set $\breve{a}$, $\triangle x = \frac{1}{l}$ and $j = 1, 2, \ldots, l$.

This process allows to map continuous variables onto discrete values. Also, similar instances whose continuous features values are slightly different are matched. More generally, two objectives are pursued with this algorithm, the first one is efficiency during the reduction process, and the second one is no impairment of data quality in terms of significant reduction of the classifier efficacy. Taking this into account, we proceeded to select a discretization method that conforms to these objectives.

There are two widely used approaches to carry out a discretization process: (1) supervised and (2) unsupervised. Supervised methods make use of a finite set of classes to partition continuous features [16]. Otherwise, unsupervised methods do not take class labels into account, which makes them more efficient than supervised ones [16]. It is valid to highlight that in intrusion detection scenarios there are data sets composed by different type of attacks (classes). For example, the KDD'99 [17] data set, which is considered a reference and widely used in intrusion detection tasks, has 23 different types of attacks. This can affect the performance of supervised techniques. An experiment on how a supervised technique can be affected is shown in the next section.

There are some comparative works where the algorithm *k*-Means is used as an unsupervised discretization method [18]. In these studies it is concluded that *k*-Means obtains more consistent and favorable results than the other unsupervised methods evaluated. Also, *k*-Means preserves the original distribution of the feature, which makes the results similar to those obtained by the supervised methods.

Another aspect that favors the use of *k*-Means over other unsupervised methods is its efficiency. The time complexity of *k*-Means, for all N instances, is given by $\mathcal{O}(N * |A| * k * i)$, where $|A|$ represents the cardinality of the feature set, k is the number of clusters and i is the number of iterations required for the algorithm to converge. This order of complexity allows *k*-Means to achieve superior performance, in terms of efficiency, to other unsupervised methods [18]. Considering that efficiency is a fundamental aspect for the proposed algorithm, *k*-Means was selected to generate the continuous feature labels. In this sense, given a value k, the algorithm follows four main steps:

1. Divide the data set into k nonempty subsets.
2. The centroid is computed for each subset.
3. Each element is assigned to the closest subset.
4. When no more assignments are possible, return to step 2. This is repeated until convergence is achieved.

To make this process faster, the Create_Labels_k-Means(k) function, used to generate labels with *k*-Means, was executed in parallel (see line 1 of Algorithm 2). The number of processes that are executed will be limited by the number of processors available on the computer used. Applying data parallelism, it was defined that each processor was going to perform the discretization process on different features. With the processes running, the set of values $\breve{a}_{i,D}$ is added to the task queue, which takes the ith feature a_i of continuous type, such that $a_i \in A$, where A represents the set of features that make up D (see lines 2–6 of Algorithm 2). Starting from the set of values $\breve{a}_{i,D}$ that a_i can take, a restriction is added for the value of k (see Eq. (6)).

Algorithm 2: Label_Generator(D, k)

Entrada: D: *Training data set*, k: *number of subsets*
Salida: T_L: *Label dictionary*

```
1 multiprocessing.Init(Create_Labels_k-Means(k),taskQueue,cpu.Count)
2 foreach i in range(1,|A| + 1) do
3 |   if a_i.type == continuous then
4 |   |   multiprocessing.workQueue.put(ă_{i,D}, T_L[a_i])
5 |   end
6 end
7 multiprocessing.Wait.taskQueue.empty()
8 multiprocessing.Wait.all_Processes.stop()
9 return T_L
```

$$k = \begin{cases} |\breve{a}_{i,D}| & if\ |\breve{a}_{i,D}| < k \\ k & otherwise. \end{cases} \tag{6}$$

Once the labels for $\breve{a}_{i,D}$ are generated, they are saved in the dictionary T_L using the feature a_i as a key. Every time a processor ends up with a set of values $\breve{a}_{i,D}$ it starts with the next one in the queue (see Fig. 4).

After processing the features in A, the proposed algorithm waits for all the tasks added to the queue to be solved (see line 7 of Algorithm 2). When the task queue is empty, it is verified that no process is executing any task (see line of 8 Algorithm 2). In the case that a process is busy, the algorithm waits for it to finish and then stops. Finally, on line 9 the Algorithm 2 returns the T_L dictionary that contains the labels associated with the value ranges of each continuous feature.

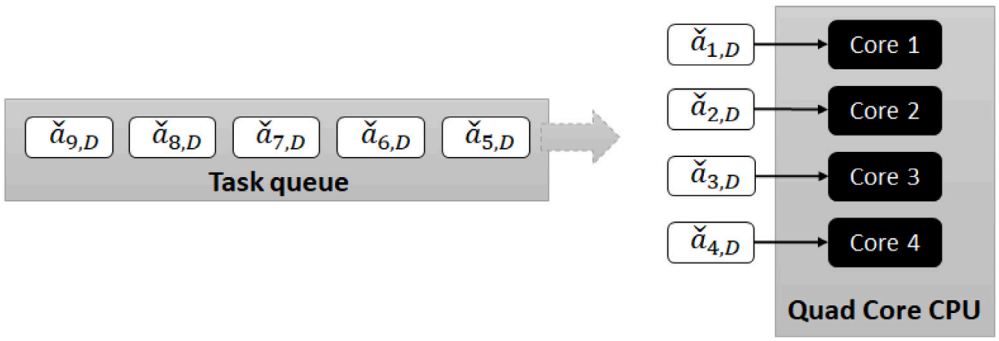

Fig. 4. Data parallelism used in FIRA.

The next stage that takes place in FIRA is relabeling, which consists of replacing the values of the continuous features with their corresponding labels (see line of 2 Algorithm 1). For this, the label dictionary T_L obtained in the previous stage and the data set D are used. The direct mapping of a continuous value v_j with its corresponding label x is given by Eq. (7), where $\sigma(\check{a}_{i,D})^x$ is the xth set of value ranges computed by k-Means for $\check{a}_{i,D}$. The result is a relabeled data set D_L that contains the same instances as D except that instead of the continuous values, there are their corresponding labels.

$$\omega(v_j) = x \mid v_j \in \sigma(\check{a}_{i,D})^x. \tag{7}$$

Finally, the instance reduction is performed on the relabeled data set D_L (see line of 3 Algorithm 1). The duplicated removing stage, as the name suggests, consists of eliminating the duplicate instances of D_L. An instance is considered duplicate if there is at least one other instance with the same class and feature values. In this way, those instances that were similar and became the same after the relabeling process will be represented by a single instance. As shown in the Algorithm 3, to eliminate duplicate instances, an iteration over D_L is performed, where each instance n, such that $n \in D_L$, is searched in the data dictionary T_D (see lines 1–5 of Algorithm 3). If n is not found, it is inserted as a key in T_D (see line 3 of Algorithm 3). After iterate over D_L, the instances that make up the keys of T_D are assigned to the reduced data set $\check{D}$ (see line 6 of Algorithm 3). The final result of FIRA is a reduced training set $\check{D}$ ready to be used by a classifier to build the classification model (see line 4 of Algorithm 1).

Algorithm 3: Duplicated_Removing(D_L)

Entrada: D_L: *Relabelled data set*
Salida: $\check{D}$: *Reduced data set*

```
1 foreach n in D_L do
2     if not T_D.has_key(n) then
3         T_D.new_key(n)
4     end
5 end
6 Ď ← T_D.keys()
7 return Ď
```

Fig. 5 shows the FIRA workflow diagram. First, multiple Label_Generator processes run in parallel, one process for each available CPU core. The Label_Generator process includes the k-Means algorithm as part of the strategy to group the closest continuous values and associate them with a single label. The result of this process is a dictionary of labels L, which is used in the relabeling stage to replace continuous values with their corresponding labels. Finally, the removal of duplicate instances is performed obtaining the reduced data set $\check{D}$.

The novelty of this work lies mainly in the heuristic used that involves a sequence of processes to achieve a higher performance than that reported by other algorithms in the literature. The label generation, the relabeling process and subsequently removing duplicate instances allow to identify and remove similar instances without the need to calculate a similarity measure between each pair of instances in the data set. This characteristic offers efficiency, in terms of execution time, higher than that achieved by other algorithms reported in literature, without significantly affecting the efficacy, in terms of accuracy, of the classifiers. The following section shows the experimental results that validate this claims.

4. Experiments and results

To evaluate the performance of the algorithm introduced in this work, a computer equipped with a 3.2 GHz Intel Quad-Core processor and 8 Gb of RAM memory was used. The experiments were carried out on two data sets: KDD'99 and CDMC 2013.

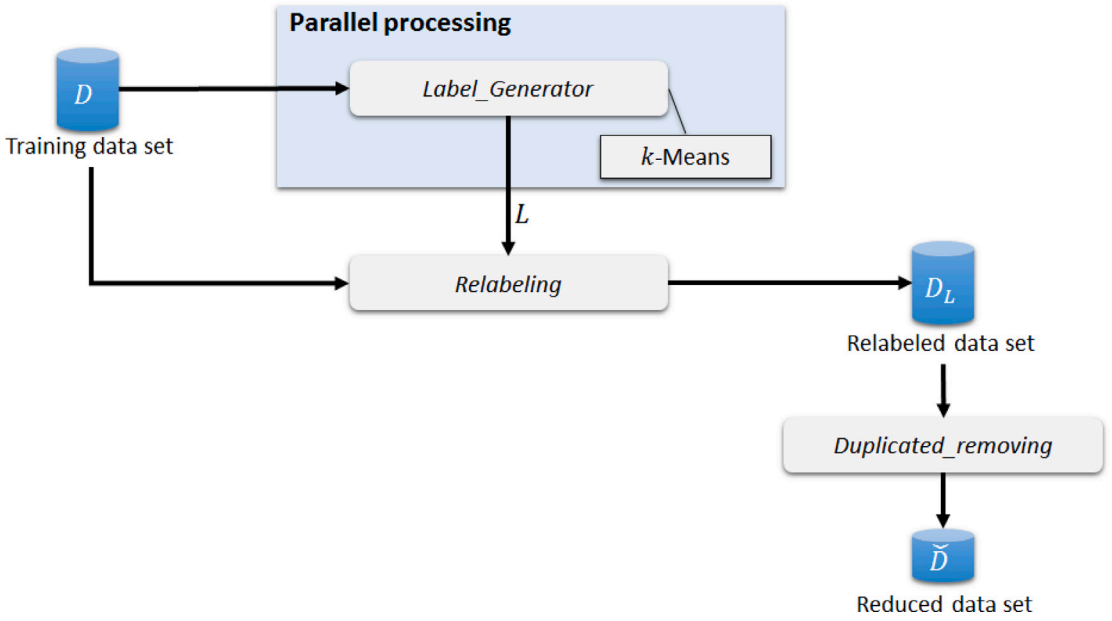

Fig. 5. FIRA workflow diagram.

KDD'99 [17] is considered a reference data set and has been used extensively for the evaluation of intrusion detection proposals [19]. This data set contains a wide variety of simulated intrusions on a military network. The training data set has 22 different types of intrusions, plus the "normal" class. The test data set contains 37 intrusion types, plus the "normal" class. All these intrusions are associated with 4 categories: (1) Probe, (2) DoS, (3) U2R and (4) R2L.

As in the reviewed works, all instances in these experiments are classified into two types: "attack" and "normal". The training set contains is 494021 instances, while the test set contains 311029. Each instance is made up of 41 features, 9 of which are discrete and 32 are continuous.

A statistical analysis of the KDD'99 data set carried out in [20], revealed some drawbacks that affect the performance of several intrusion detection systems. This fact led to propose a new data set called NSL-KDD [20], which consists of selected records of the complete KDD'99 data set. The full NSL-KDD training collection with 125973 instances and the full NSL-KDD testing collection with 22544 instances were used in our experiments. Like KDD'99, the NSL-KDD data set contains 41 features out of which 9 are discrete and 32 are continuous. Also, all the instances were classified into two types, "normal" and "anomaly".

In the case of CDMC 2013 [21], it is a data set created from a real intrusion detection system. Without loss of generality, CDMC 2013 was divided into a training set with 40000 instances and a test set with 37959 instances. Each instance is made up of 7 numeric features, in addition to the feature that defines the class.

KNN and SVM classifiers have been widely used in intrusion detection tasks, reporting good results in terms of efficacy [22]. In addition, KNN have been found to be an efficient technique, during training stage, and useful for intrusion identification due to better generalization ability in complex domains. However, the efficiency of the KNN classification process is often affected when the classification model is built from large volumes of data [23]. On the other hand, the complexity of the time taken to train a classifier is also a major concern, because many supervised learning algorithms, such as SVM, require very expensive calculations in the presence of large data sets [23]. Taking into account that the complexity of both classifiers is dependent on the volume of data used during training, it was decided to choose both classifiers to evaluate the quality of the reduced data sets.

The efficacy of the classifiers is estimated with the accuracy (Acc) measure defined in Eq. (8), where T^+, T^-, F^+ and F^- represent true positives, true negatives, false positives, and false negatives, respectively.

$$Acc = \frac{T^+ + T^-}{T^+ + T^- + F^+ + F^-} * 100. \tag{8}$$

The first experiment aims to show how the use of a discretization technique different from the one conceived in the proposed algorithm can influence the performance of FIRA. For this, k-Means was replaced by the Minimum Description Length (MDL) [24], which is a supervised discretization technique widely used in intrusion detection scenarios [16]. The MDL-based discretization technique does not require a k value to be defined. In the case of using k-Means, it is possible to define such value, and thus, adjust the size of the data set to be obtained; the smaller the value of k, the more the data set is reduced, and vice versa. Given the above, k-Means was evaluated for different values of k. These values were chosen with the purpose of obtaining a reduced data set of a size similar to that obtained with MDL. This allowed to make the following comparisons.

For the KDD'99 data set, the average number of labels generated by FIRA using the MDL-based discretization technique is 10. By using this number with k-Means, the proposed FIRA ($k = 10$) algorithm obtains a smaller data set compared to the previous one,

Table 1
Efficacy achieved on the KDD'99 data set using FIRA with different discretization techniques.

Algorithm	SVM	KNN	Instances
FIRA (MDL)	88.14	88.09	40262
FIRA ($k = 10$)	87.28	87.37	24132
FIRA ($k = 14$)	88.20	88.17	38431

Table 2
Efficacy achieved on the NSL-KDD data set using FIRA with different discretization techniques.

Algorithm	SVM	KNN	Instances
FIRA (MDL)	70.94	74.68	8415
FIRA ($k = 11$)	70.12	74.07	7334
FIRA ($k = 14$)	71.23	74.79	8221

Table 3
Efficacy achieved on the CDMC 2013 data set using FIRA with different discretization techniques.

Algorithm	SVM	KNN	Instances
FIRA (MDL)	94.90	95.01	1207
FIRA ($k = 18$)	94.72	94.83	821
FIRA ($k = 20$)	95.05	95.12	1071

with approximately 50 % fewer instances (see Table 1). This fact leads to classifiers obtaining less efficacy using FIRA with *k*-Means. However, using the FIRA ($k = 14$) algorithm, the reduced data set is almost similar in size to that obtained with MDL, although slightly smaller. Under these conditions, the classifiers achieved their best efficacy.

The MDL discretization technique generates 11 labels for the NSL-KDD data set (see Table 2). When using FIRA with *k*-Means ($k = 11$), a behavior similar to that observed in the previous comparison is appreciable. That is, FIRA with *k*-Means ($k = 11$) obtains a smaller data set than using MDL (approximately 1000 fewer instances), which negatively affects the efficacy of the classifiers. By increasing the value of k to 14, the number of instances in the reduced set approaches, but does not exceed, the number obtained with the MDL technique. However, the efficacy reported by the classifiers using the data set reduced by FIRA with *k*-Means ($k = 14$) exceeds that obtained by FIRA with MDL.

For CDMC 2013 data set (see Table 3), the average number of labels generated using MDL-based discretization technique is 18. In this case, a comparison similar to the previous one can be established, obtaining the best results using FIRA with *k*-Means ($k = 20$).

As could be observed in the previous experiments, using *k*-Means ($k = 20$) it is possible to obtain a smaller data set than using MDL. Even under these circumstances, the reduced data set with *k*-Means provides greater efficacy to the classifiers. The selected value of k could be larger, but this would imply a greater number of instances in the reduced data set, which would result into less efficiency. Therefore, for each data set, different maximum values of k were defined for *k*-Means, which allowed to obtain a data set smaller than the obtained by MDL, and still provide greater efficacy to the classifiers.

Taking into account that the KDD'99 data set can be used for a multiclass classification problem, the following experiment was performed by only using this data set. The experiment consists of evaluating the efficiency of both discretization techniques in binary classification and multiclass classification problems. For this, three different configurations of KDD'99 were used, according to the number of classes:

1. KDD'99 with 2 classes (Normal and Attack).
2. KDD'99 with 5 classes (Normal, Probe, DoS, U2R and R2L).
3. KDD'99 with 23 classes (Normal and 22 different types of attack).

As shown in Fig. 6, when processing the KDD'99 data set with a low number of classes (2 and 5 classes), the MDL discretization technique shows better execution time than *k*-Means. However, the *k*-Means input parameters have an influence on its performance. For example, the parameter for the number of iterations was the default one, which is 200 iterations. If this value is reduced, the execution time of *k*-Means is also reduced, but more experimental work would be needed to define an optimal value of iterations that do not subsequently compromise the quality of the generated clusters. On the other hand, when the KDD'99 data set is processed as a more complex multiclass classification problem (23 different classes), more in line with a real-world scenario, it can be observed how the performance of MDL is affected: here *k*-Means shows a linear behavior that does not affect its execution time. This shows how a supervised discretization technique can affect its efficiency when processing data sets with a high number of classes.

Taking into account that FIRA using *k*-Means provides a better efficacy to classifiers than using MDL, even with a smaller data set, and that *k*-Means shows higher efficiency than MDL in data sets with a high number of classes, it is feasible to use *k*-Means as a discretization technique in the proposed algorithm.

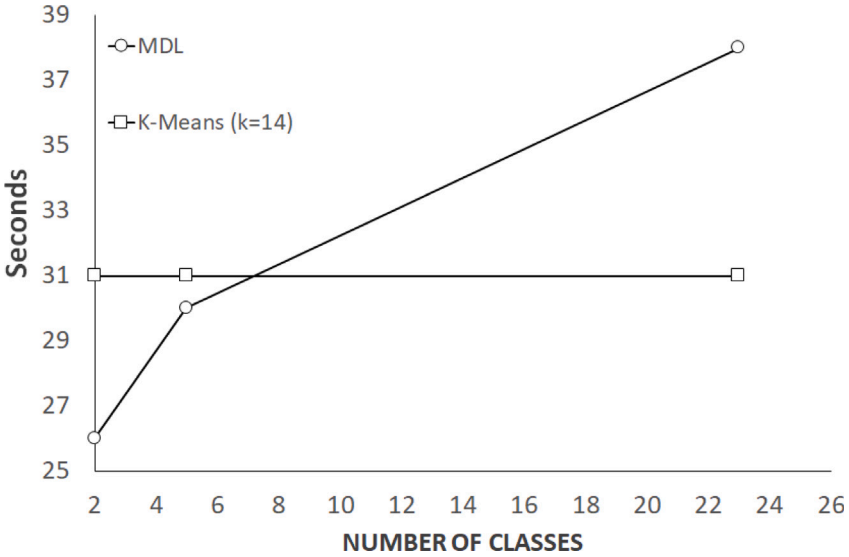

Fig. 6. Runtime reached by FIRA with different discretization techniques over KDD'99 data set.

In the following experiments the performance of FIRA is evaluated and compared regarding to the results achieved by the instance selection algorithms proposed in [10,12–14] that were analyzed in Section 2.

Before evaluating the efficiency of the proposed algorithm, its time complexity was estimated. FIRA consists of three fundamental steps. The first step consists of the generation of labels (f_1), where the k-Means algorithm is used. As described in the previous section, the time complexity of k-Means, for all N instances, is $\mathcal{O}(N * |A| * k * i)$ [25]. Considering that k-Means is used to discretize the values of continuous features, in the worst case, its time complexity is given by $\mathcal{O}(|A_D| * k * i)$, where $|A_D|$ is the cardinality of the feature set with more different values in D. In the relabeling stage (f_2) each instance is traversed, replacing the feature values with their corresponding labels, by performing a logarithmic search in a dictionary. Assuming that each feature takes the maximum possible values, in the worst case, its complexity is given by $\mathcal{O}(N * \log |A_D|)$. Duplicated removing ($f_3$) is the most expensive of the three stages. The process consists of iterating through each instance and checking if it has already been added to a hash table. In general, this search has a complexity of $\mathcal{O}(1)$, but in the worst case, it can reach the time complexity of $\mathcal{O}(N)$. Therefore, in the worst case, the order of time complexity is $\mathcal{O}(N^2)$. Considering the previous analysis, the time complexity of each step would be defined as follows: $f_1 \in \mathcal{O}(|A_D| * k * i)$, $f_2 \in \mathcal{O}(N * \log |A_D|)$ and $f_3 \in \mathcal{O}(N^2)$. As shown in Eq. (9), when applying the sum rule, the complexity that defines the FIRA algorithm is $\mathcal{O}(N^2)$.

$$f_1 + f_2 + f_3 \in \mathcal{O}(max(|A_D| * k * i, N * \log |A_D|, N^2)) = \mathcal{O}(N^2). \tag{9}$$

The computational complexity of the compared algorithms were not given in the reviewed papers. However, by analyzing the pseudocode and the explanations given by the authors it was possible to infer the time complexity of the proposals. The algorithms proposed in [10,12–14], in the worst case, have the same time complexity as FIRA ($\mathcal{O}(N^2)$), however the execution times are different.

Fig. 7 shows the execution time (in seconds) taken by each algorithm to process different volumes of the KDD'99 data set, represented by the number of instances. Note that FIRA was executed for different values of k. As explained above, these values were chosen with the purpose of obtaining reduced data sets of similar size to those obtained by the analyzed algorithms. In practice, it is possible to rely on measures that allow obtaining an optimal value of k, that is, a value of k that minimizes the intra-cluster distance while maximizing the inter-cluster distance. The use of the Silhouette coefficient is usually common for these cases [26]. However, if you want to directly influence the size of the data set, you can define the value of k manually.

As can be seen in Fig. 7, the FIRA execution time and the value of k are directly proportional. This is because k defines the maximum number of labels that are generated for each feature. With more labels, there will be fewer duplicate instances. Therefore, the fewer duplicate instances there are in the relabeled data set, the greater the execution time of the duplicated removing stage, since a greater number of instances are added to the hash table. However, the FIRA execution time for any of the k values used is considerably less than in the [10,13,14] proposals.

In the case of the [12] proposal, it is fair to compare it with the execution time reported by the FIRA algorithm for $k = 4$, since with this value FIRA obtains a reduced data set of similar size to the one obtained by [12]. In this sense, FIRA is up to five times faster than the [12] algorithm. In general, it can be observed that regardless of the value defined for k, FIRA proved to be more efficient than the other proposals evaluated. Furthermore, it can be seen that FIRA scales linearly with respect to k.

Although the time complexity of FIRA and some reported algorithms is the same, FIRA achieves the best results in terms of efficiency. The main reason is that the most expensive stage of FIRA (duplicated removing) always processes a relabeled data set

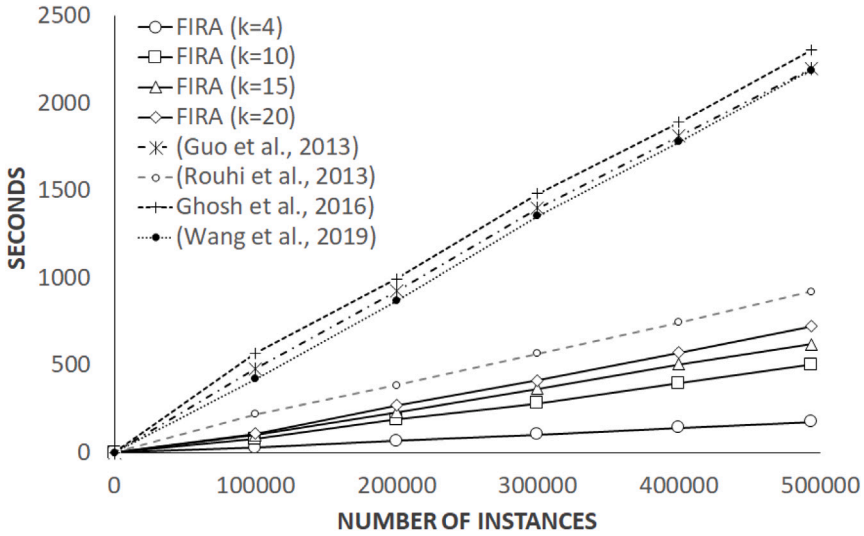

Fig. 7. Runtimes when processing different volumes of KDD'99.

Table 4
Efficacy achieved on the KDD'99 data set.

Algorithm	Classifier	*Acc*	Instances
Original	KNN	92.81	494021
[12]		84.87	2768
[13]		88.91	61758
[14]		88.03	48854
[10]		88.97	61202
FIRA ($k = 4$)		85.04	2621
FIRA ($k = 10$)		87.37	24132
FIRA ($k = 15$)		88.21	41736
FIRA ($k = 20$)		89.17	60375
Original	SVM	92.30	494021
[12]		84.23	2768
[13]		88.97	61758
[14]		88.12	48854
[10]		89.02	61202
FIRA ($k = 4$)		84.85	2621
FIRA ($k = 10$)		87.28	24132
FIRA ($k = 15$)		88.25	41736
FIRA ($k = 20$)		89.05	60375

with many duplicate instances. This requires a number of instances less than N to be stored in the hash table, which prevents a time complexity of order N^2 (worst case) being reached at this stage.

Once the data had been reduced, an experiment focused on evaluating the quality of the new data set was performed. For this, the KNN and SVM classifiers were trained with the data sets obtained by each instance selection algorithm, as well as with the original data set. Once the classification models for each data set were built, they were evaluated with their respective test data sets. The results achieved, in terms of efficacy, are discussed below.

In Table 4 it can be seen how FIRA, with $k = 20$, reduces the data set to a greater extent than the proposals of [10,13], and still provides greater efficacy than both proposals in the classifiers. On the other hand, using FIRA with $k = 15$, the reduction of the data set is greater than that of [14] algorithm; however, the efficacy achieved by the classifiers trained with the data set reduced by FIRA ($k = 15$) exceeds the reported efficacy using the data set reduced by the [14] proposal. A similar comparison can be done for FIRA ($k = 4$) and the [12] proposal.

Note, that the number of instances shown by each algorithm in Table 4 is the same for both classifiers (KNN and SVM). This is because each algorithm reduces the data set only once and the reduced data set is evaluated in each classifier separately, obtaining different results.

The results achieved on the NSL-KDD and CDMC 2013 data sets are shown in Tables 5 and 6 respectively, where similar results to those discussed above can be seen.

In the Tables 4–6, it can be observed how efficacy improves as the value of k gets larger. This improvement is due to the fact that when k increases, more labels are created for each continuous feature, which causes that, after the relabeling process, the number

Table 5
Efficacy achieved on the NSL-KDD data set.

Algorithm	Classifier	*Acc*	Instances
Original	KNN	79.36	125973
[12]		73.81	7303
[13]		75.90	11417
[14]		74.89	8610
[10]		75.99	11692
FIRA ($k = 4$)		65.73	2006
FIRA ($k = 10$)		73.88	7113
FIRA ($k = 15$)		74.92	8515
FIRA ($k = 20$)		76.02	11046
Original	SVM	75.39	125973
[12]		69.87	7303
[13]		72.80	11417
[14]		71.60	8610
[10]		72.86	11692
FIRA ($k = 4$)		60.61	2006
FIRA ($k = 10$)		69.91	7113
FIRA ($k = 15$)		71.67	8515
FIRA ($k = 20$)		72.95	11046

Table 6
Efficacy achieved on the CDMC 2013 data set.

Algorithm	Classifier	*Acc*	Instances
Original	KNN	96.57	40000
[12]		88.17	153
[13]		95.01	1259
[14]		94.47	813
[10]		95.10	1300
FIRA ($k = 4$)		88.46	129
FIRA ($k = 10$)		92.48	387
FIRA ($k = 15$)		94.78	679
FIRA ($k = 20$)		95.12	1071
Original	SVM	96.42	40000
[12]		87.94	153
[13]		94.82	1259
[14]		94.29	813
[10]		95.01	1300
FIRA ($k = 4$)		88.22	129
FIRA ($k = 10$)		92.41	387
FIRA ($k = 15$)		94.61	679
FIRA ($k = 20$)		95.05	1071

of duplicate instances decreases. As there are fewer duplicates, the original data set is reduced to a lesser extent, conserving more relevant information which is used by the classifier to achieve better efficacy.

Fig. 8 shows the time taken by KNN to build the classification model on different data sets. As can be seen in Fig. 8(a), the time taken to build the classification model is reduced by more than half when FIRA is used on the KDD'99 data. However, the improvement in terms of time is practically imperceptible, since the KNN training stage is very fast, being able to process half a million instances in less than a second. A similar situation can be seen in Figs. 8(b) and 8(c), where the difference in time is even more imperceptible, since NSL-KDD and CDMC 2013 data sets are smaller than KDD'99.

In case of SVM, there is a considerable improvement over the original KDD'99 data set (see Fig. 9(a)). FIRA is able to reduce the time taken by SVM to build the classification model with the original data set by more than 82 %. In Fig. 9(a), it can be observed that the time consumed by SVM using the [10,13] proposals is similar to that reported using FIRA ($k = 20$), only with the latter is 15 s faster. Furthermore, if the execution time of FIRA ($k = 20$) and its efficacy are taken into account, it would be more feasible to use such an algorithm, instead of the [13] or [10] proposals. A similar comparison can be done with FIRA ($k = 15$) and the algorithm proposed by [14], only in this case the difference in time is more visible, thus the SVM classifier is 196 s faster with the data set reduced by FIRA ($k = 15$). With the [12] algorithm, SVM reports practically the same time as using FIRA ($k = 4$). However, the results reported in terms of efficacy and size of the reduced data set presented in Table 4 make the use of FIRA ($k = 4$) more feasible. The same behavior described for KDD'99 data set is observed in Fig. 9(b) when processing the NSL-KDD data set. On the CDMC 2013 data set, the improvement in time is not perceptible (see Fig. 9(c)). This is due to the fact that the data size is very small, so the classifier can process it immediately.

To find out if the achieved results by the evaluated classifiers are significantly different, in terms of efficacy, a statistical test was carried out. Taking into account that the reported efficacy follows a continuous uniform distribution, the nonparametric Friedman

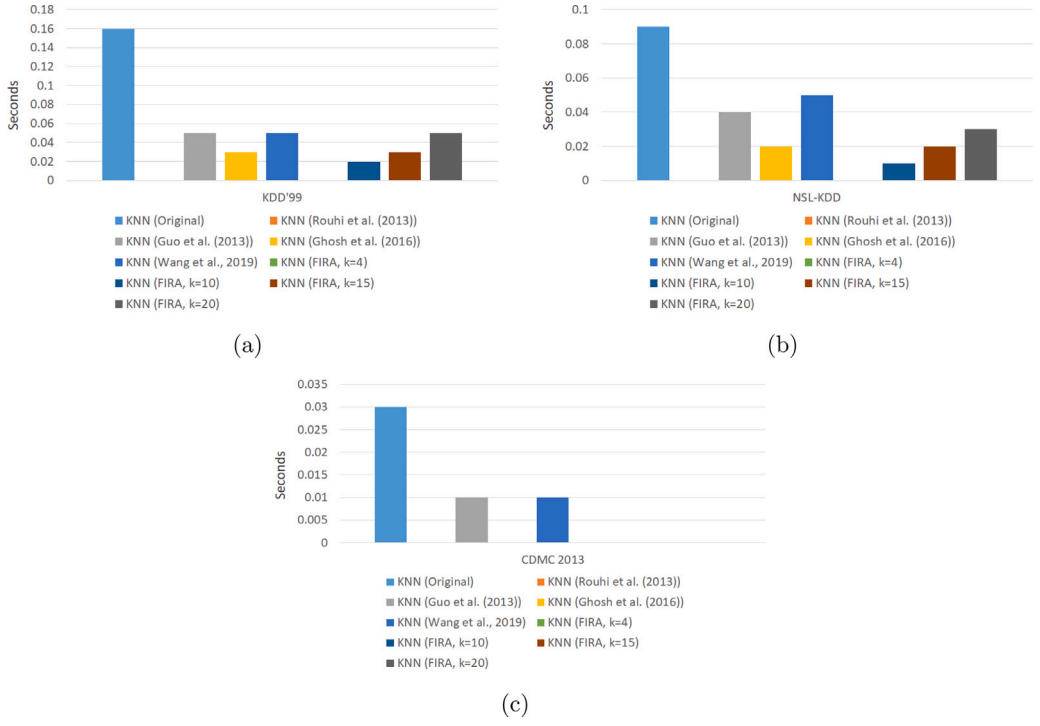

Fig. 8. Time taken by the KNN classifier to build its classification model using different data sets.

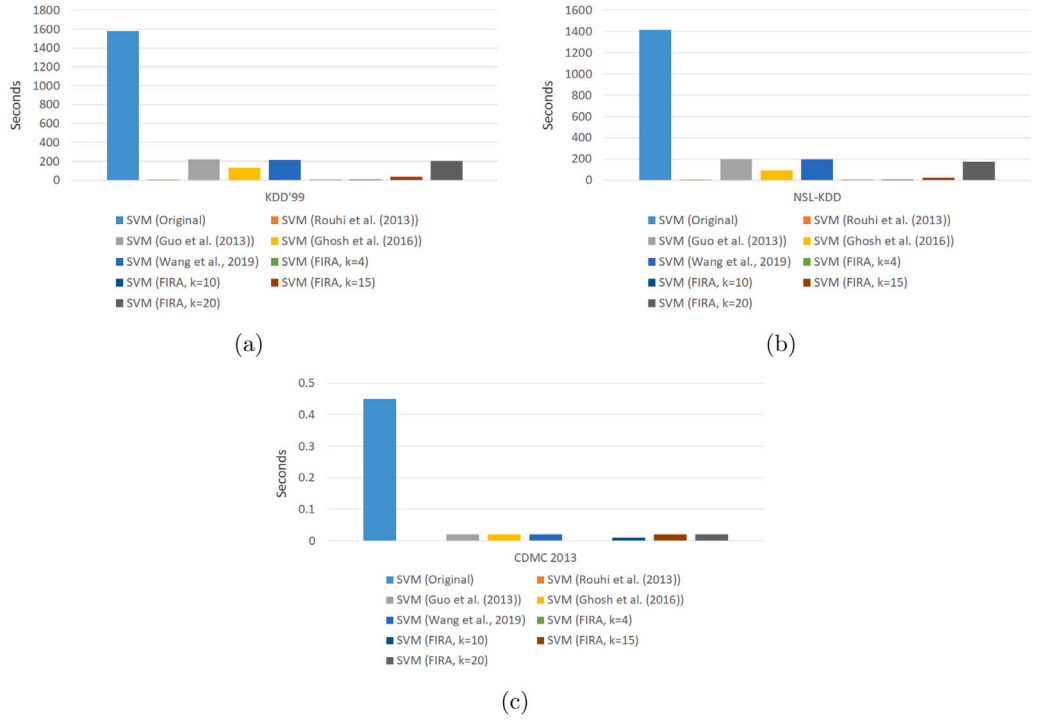

Fig. 9. Time taken by the SVM classifier to build its classification model using different data sets.

test was applied [27]. Subsequently, to determine which proposals show a statistically similar behavior, the Bergmann–Hommel *post-hoc* procedure was performed [28].

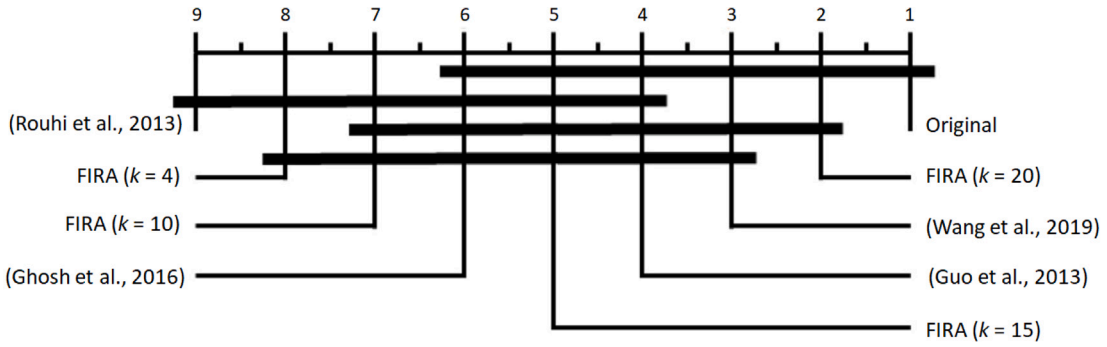

Fig. 10. Critical difference diagram with a statistical comparison of the efficacy achieved by the classifiers using different algorithms for instances selection.

The statistical results are presented in a critical difference diagram (see Fig. 10). The critical difference diagram shows in a compact way the order of the classifiers based on the ranking obtained by the Friedman test, the magnitude of the differences between them, and the significance of these differences. The position of the classifiers in the segment represents their value in the ranking, with the classifier furthest to the right being the one with the best result. If two or more classifiers share a thick line it indicates that they show statistically similar behavior.

As seen in Fig. 10, when the classifiers use the original data set, they obtain the best results. When low values for k are used in FIRA, the data set is reduced to a greater extent, which implies a greater loss of information that leads to a significant impact on efficacy. However, it can be seen that there are no statistically significant differences between the achieved results by the classifiers with the original data set and with the data set reduced by FIRA using a value of $k = 20$ and $k = 15$.

The previous result shows that using FIRA, it is possible to considerably reduce the original data set, achieving better efficacy and efficiency than other reported proposals, without significantly affecting the efficacy regarding to that obtained with the original data set.

5. Conclusions

Based on observations performed on collected data from intrusion detection scenarios, it was possible to propose an algorithm to efficiently reduce the data set, without significantly affecting the efficacy during the classification process.

The instance selection algorithm proposed in this work achieves satisfactory results, even in large-volume data sets, where other proposals are affected, in terms of efficiency. This is due to an effective combination of heuristics that guarantee the consistency of the algorithm. Using k-Means to group the values of continuous features without affecting their original distribution, allows to efficiently identify instances that are somewhat similar. In this way, the similar instances after the relabeling process become the same, which facilitates the reduction process since with the elimination of duplicate instances it is possible to quickly reduce the data set.

Implementing instance reduction strategies that ensure acceptable performance in these scenarios is often costly in terms of time and resources. Taking into account that FIRA showed good performance on a standard PC and also achieved superior results than other proposed algorithms, in terms of efficacy and efficiency, it is concluded that its use in real scenarios is feasible, where it can provide a considerable improvement in the performance of intrusion detection systems.

However, as shown in the experiments, increasing the volume of data negatively affects the efficiency of all the algorithms evaluated, including FIRA. We intend to address this scalability problem in future work with the extension of FIRA to a distributed environment that allows an efficient handling of large-scale data processing (Big Data).

CRediT authorship contribution statement

Vitali Herrera-Semenets: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft. **Raudel Hernández-León:** Supervision, Project administration, Formal analysis. **Jan van den Berg:** Writing – review & editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.compeleceng.2022.107963.

References

[1] Oleg Kupreev, Ekaterina Badovskaya, Alexander Gutnikov. Ddos attacks in Q2 2020. 2020, Available on https://Securelist.Com/Ddos-Attacks-in-Q2-2020/98077/.
[2] Mohammad Abdul Salam, Pradhan Manas Ranjan. Machine learning with big data analytics for cloud security. Comput Electr Eng 2021;96:107527.
[3] Rambabu Kalathiripi, Venkatram N. Ensemble classification using traffic flow metrics to predict distributed denial of service scope in the internet of things (IoT) networks. Comput Electr Eng 2021;96:107444.
[4] García Salvador, Luengo Julián, Herrera Francisco. Data preprocessing in data mining. Springer; 2016.
[5] Baraneetharan E. Role of machine learning algorithms intrusion detection in WSNs: a survey. J Inf Technol 2020;2(03):161–73.
[6] Panigrahi Ranjit, Borah Samarjeet, Bhoi Akash Kumar, Ijaz Muhammad Fazal, Pramanik Moumita, Jhaveri Rutvij H, et al. Performance assessment of supervised classifiers for designing intrusion detection systems: a comprehensive review and recommendations for future research. Mathematics 2021;9(6):690.
[7] Qu Xiaofei, Yang Lin, Guo Kai, Ma Linru, Sun Meng, Ke Mingxing, et al. A survey on the development of self-organizing maps for unsupervised intrusion detection. Mob Netw Appl 2021;26(2):808–29.
[8] Herrera-Semenets Vitali, Pérez-García Osvaldo Andrés, Hernández-León Raudel, van den Berg Jan, Doerr Christian. A data reduction strategy and its application on scan and backscatter detection using rule-based classifiers. Expert Syst Appl 2018;95:272–9.
[9] Sharafaldin Iman, Lashkari Arash Habibi, Hakak Saqib, Ghorbani Ali A. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In: 2019 International carnahan conference on security technology. IEEE; 2019, p. 1–8.
[10] Wang Qiuhua, Ouyang Xiaoqin, Zhan Jiacheng. A classification algorithm based on data clustering and data reduction for intrusion detection system over big data. KSII Trans Internet Inf Syst 2019;13(7).
[11] Baldini Gianmarco, Hernandez-Ramos Jose L. An intrusion detection system implemented with instance selection based on locality sensitive hashing for data reduction. In: European wireless 2021; 26th European wireless conference. VDE; 2021, p. 1–6.
[12] Rouhi Rahimeh, Keynia Farshid, Amiri Mehran. Improving the intrusion detection systems performance by correlation as a sample selection method. J Comput Sci Appl 2013;1(3):33–8.
[13] Guo Chun, Zhou Ya-Jian, Ping Yuan, Luo Shou-Shan, Lai Yu-Ping, Zhang Zhong-Kun. Efficient intrusion detection using representative instances. Comput Secur 2013;39:255–67.
[14] Ghosh Partha, Saha Akash, Phadikar Santanu. Penalty-reward based instance selection method in cloud environment using the concept of nearest neighbor. Procedia Comput Sci 2016;89:82–9.
[15] Dai Bi-Ru, Hsu Shu-Ming. An instance selection algorithm based on reverse nearest neighbor. In: Pacific-Asia conference on knowledge discovery and data mining. Springer; 2011, p. 1–12.
[16] Tsai Chih-Fong, Chen Yu-Chi. The optimal combination of feature selection and data discretization: An empirical study. Inform Sci 2019;505:282–93.
[17] Siddique Kamran, Akhtar Zahid, Khan Farrukh Aslam, Kim Yangwoo. KDD cup 99 data sets: a perspective on the role of data sets in network intrusion detection research. Computer 2019;52(2):41–51.
[18] Dash Rajashree, Paramguru Rajib Lochan, Dash Rasmita. Comparative analysis of supervised and unsupervised discretization techniques. Int J Adv Sci Technol 2011;2(3):29–37.
[19] Ring Markus, Wunderlich Sarah, Scheuring Deniz, Landes Dieter, Hotho Andreas. A survey of network-based intrusion detection data sets. Comput Secur 2019;86:147–67.
[20] Tavallaee Mahbod, Bagheri Ebrahim, Lu Wei, Ghorbani Ali A. A detailed analysis of the KDD cup 99 data set. In: Computational intelligence for security and defense applications, 2009. CISDA 2009. IEEE symposium on. IEEE; 2009, p. 1–6.
[21] Song J. CDMC2013 intrusion detection dataset. 2013, Department of Science & Technology Security, Korea Institute of Science and Technology Information (KISTI).
[22] Zhao Fei, Xin Yang, Zhang Kai, Niu Xinxin. Representativeness-based instance selection for intrusion detection. Secur Commun Netw 2021;2021.
[23] Itoo Fayaz, Singh Satwinder. Comparison and analysis of logistic regression, naïve Bayes and KNN machine learning algorithms for credit card fraud detection. Int J Inf Technol 2021;13(4):1503–11.
[24] Kononenko Igor. On biases in estimating multi-valued attributes. In: Ijcai, Vol. 95. Citeseer; 1995, p. 1034–40.
[25] Ahmed Mohiuddin, Seraj Raihan, Islam Syed Mohammed Shamsul. The k-means algorithm: A comprehensive survey and performance evaluation. Electronics 2020;9(8):1295.
[26] Dudek Andrzej. Silhouette index as clustering evaluation tool. In: Conference of the section on classification and data analysis of the polish statistical association. Springer; 2019, p. 19–33.
[27] Demšar Janez. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 2006;7(Jan):1–30.
[28] Garcia Salvador, Herrera Francisco. An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. J Mach Learn Res 2008;9(Dec):2677–94.

Vitali Herrera Semenets studied computer science at the University of Informatics Sciences in Havana. His research is focused on the application of data mining algorithms in the detection of malicious activities in telecommunications networks. He currently works as head of the data mining team at Advanced Technologies Application Center (CENATAV), Cuba.

Raudel Hernández León studied computer science at Havana University at the end of the nineties of the previous century. His research has covered many areas including data mining and pattern recognition. He currently acts as data mining researcher in Datys, a high-tech company specialized in software applications development that offers its own solutions to complex technological problems.

Jan van den Berg studied maths and physics at Delft University of Technology in the early seventies of the previous century. His research has covered many areas including computational intelligence and the development of an holistic conceptualization of cyberspace and cybersecurity. He currently acts as an emeritus professor Cyber Security both at Delft University and Leiden University, The Netherlands.