Fast and straightforward feature selection method ⋆

A case of high dimensional low sample size dataset in malware analysis

Sergii Banin[1]

Abstract Malware analysis and detection is currently one of the major topics in the information security landscape. Two main approaches to analyze and detect malware are *static* and *dynamic* analysis. In order to detect a running malware, one needs to perform dynamic analysis. Different methods of dynamic malware analysis produce different amounts of data. The methods that rely on low-level features produce very high amounts of data. Thus, machine learning methods are used to speed up and automate the analysis. The data that fed into machine learning algorithms often requires preprocessing. Feature selection is one of the important steps of data preprocessing and often takes significant amount of time. In this paper we analyze the Intersection Subtraction (IS) feature selection method that was first proposed and used on a high dimensional dataset derived from the behavioral malware analysis. In our work, we assess its computational complexity and analyze potential strengths and weaknesses. In the end, we compare Intersection Subtraction and Information Gain (IG) feature selection methods in term of potential classification performance and time complexity. We apply them to the dataset of memory access patterns produced by malicious and benign executables. As the result we found, that the features selected by IS and IG are very different. Nevertheless, machine learning models trained with IS-selected features performed almost as good as those trained with IG-selected features. IS allowed to achieve the classification accuracy of more than 99%. We also show, the IS feature selection method is faster than IG what makes it attractive to those who need to analyze high dimensional datasets.

Sergii Banin
Department of Information Security and Communication Technology, NTNU, Gjøvik, Norway, e-mail: sergii.banin@ntnu.no

⋆ The research leading to these results has received funding from the Center for Cyber and Information Security, under budget allocation from the Ministry of Justice and Public Security of Norway

1 Introduction

Today many researchers from different research areas have to deal with big amounts of data. Various statistical methods are used to process and understand data that is too big or complex for human analysis. Part of these methods are called *machine learning*: "the automatic modeling of underlying processes that have generated the collected data" [22]. Currently, machine learning is one of the most used approaches when there is a need to predict certain qualities of objects or events. Machine learning algorithms can be divided into supervised (classification and regression) and unsupervised (clustering). In this paper we focus on the classification: prediction of a *class* (type) of a sample based on its *features* (properties). Machine learning is widely used in different fields such as medicine, biology, manufacturing [24] or information security [31] [4]. In information security, machine learning is extensively used in production and research, as the amounts of data that need to be processed are enormous. Especially, machine learning is actively used for malware analysis and detection. According to AV-TEST Institute, there are more than 350,000 new malware samples detected every day [3]. The developers of the anti-virus solutions and researchers work on finding a way to detect malware without having to search through the entire database of already known malware. Moreover, they try to find methods that allow detecting previously unknown malware. The common practice is to find certain characteristics that are common to many malware samples. As the number of malware is very big and growing [3] the machine learning methods are used to deal with the emerging amount of data. Machine learning methods rely on *features*: properties of objects that are being studied. There are two main types of features that can be extracted from malware: static and dynamic. Static features are extracted directly from the malicious file without a need to launch it. Static features are relatively easy to extract, but at the same time it is easier to change them with a use of obfuscation or encryption [1]. However, malware becomes malicious only after it has been launched. The features that occur after the launch of malware are called *dynamic*, or *behavioral* features. We can divide dynamic features into high- and low-level features [6]. File and network activity, API [2] and system calls are some of the high-level features, while opcodes, memory access operations [38] or hardware performance counters are considered to be low-level ones. We name dynamic features that emerge from the system's hardware as the low-level features [5] [21] [25]. To represent a certain behavioral event with low-level features we need to record and process a significantly bigger amount of data. For example, to describe an API call on the high level we only need its name and arguments passed to it on the call. However, if we decide to record a sequence of opcodes or memory access operations invoked by the API call we'll end up with hundreds if not thousands of events. In this paper, we address a problem that arises from the number of low-level features one needs to record and process while doing dynamic malware analysis.

While machine learning provides good opportunities for automation and analysis, the data that is used by machine learning algorithms has to be preprocessed. Various methods of data preprocessing are described in the literature: discretization of continuous features, attribute binarization, the transformation of discrete features

into continuous, dimensionality reduction and so on [22]. The first three of the aforementioned methods are mostly used when the chosen machine learning algorithm works only with a certain type of data. For example, the Naive Bayes classifier needs discrete data to provide a useful outcome. On its turn, dimensionality reduction is often needed, when the amount of features in the dataset is too big. Having too many features can result in increased model training times and model overfitting. There are several ways to reduce dimensionality: feature subset selection, feature extraction and principal components analysis (PCA) [22]. Feature extraction is aimed at finding a set of new features that are constructed as a function of original features. On its turn, PCA finds a new coordinate system with a focus on making the axes aligned with the highest variance of the data. These methods, however, make it harder to analyze the results achieved by the machine learning model: it is sometimes important to understand which features contribute the most towards the classification performance of a model. In such cases, in order to reduce the dimensionality, one may apply feature (subset) selection. With feature selection it is possible to select a certain amount of *best* features based on a certain feature quality measure while keeping the original features intact.

Feature selection is aimed at the dimensionality reduction. Ironically, when the amount of features becomes too big (for example millions as in [8] or [5]) the feature selection becomes a very computationally intense task. The datasets where the number of features is much bigger than the number of learning samples are called High-dimensional low (small) sample size (HDLSS/HDSSS) datasets. Sometimes there are so many features [8], that commonly used machine learning packages simply can not handle such datasets. Storing such a dataset in the single file or database table becomes a problem as well. Thus, the use of the common machine learning packages becomes impossible since they require data to be stored in one piece. On its turn, developing and implementation of a custom machine learning package can take more time than actual data collection and be a hard task for the researchers that don't have enough expertise in software development.

In this paper we focus on the feature selection method that was developed and used in [8] to detect malware based on the memory access patterns. In [8] the dataset contained almost six millions of binary features and 1204 samples divided into two classes. The features represented sequences of memory access operations generated by malicious and benign software. The feature took value *1* if it was generated by a sample, and *0* if not. Utilized feature selection method was aimed at removing those features, that are *present* (take value 1) in the samples of both classes. Thus, it is named Intersection Subtraction (IS) feature selection method. This method helped authors of [8] to reduce feature space from 6M of features to 800. With the use of selected features, it became possible to train a classification model that achieved 98% classification accuracy for the two-class dataset. In this paper, we provide an additional analysis of the IS feature selection method and discuss its advantages and disadvantages. We also compare its performance with an Information Gain [22] feature selection method in a similar malware detection problem. We run our tests on the newer and larger dataset of malicious and benign executables. We show how machine learning models trained with features selected by IS feature selection per-

form compared to those selected by IG.

The remainder of the paper is arranged as follows. In Section 2 we describe the problem and provide an overview of related articles. In Section 3 we describe the IS feature selection method, theoretically assess its strengths and weaknesses and explain the context in which IS might be used. In the Section 4 we describe our experimental setup, compare feature sets selected by IS and IG, and train machine learning algorithms with the use of selected features. In Section 5 we discuss our findings and outline the future work. In the last Section 6 we summarize our findings and provide conclusions.

2 Background

In this section, we describe the problem area and provide an overview of the papers related to HDLSS datasets and feature selection.

2.1 Problem description

While talking about the optimal size of the dataset to be used in machine learning model training, different authors consider different dataset sizes to be optimal. The size of the dataset consists of a number of samples and features. In various sources [26] [15] one can find suggestions, that a minimal amount of samples for training should be between 50 and 80, while 200 and more samples are expected to bring increased accuracy and significantly smaller error rates. Other authors have shown that it is important to have at least 20 to 30 samples per class [11]. When talking about the number of features it is generally considered, that the fewer features there are in the dataset - the better it is for machine learning algorithm [8] [5] [7] [22]. Some authors advise utilizing *the rule of 10*: in order to train a model with a good performance, one needs to have ten times more samples than the number of features [23]. However, in some cases, the number of features can be significantly higher than the number of learning samples. This may happen due to the context of the research and the nature of data. For example, in [8] the authors describe a novel malware detection approach. They record memory access operations performed by malicious and benign executables, split them into n-grams of various sizes and use those n-grams as features for training the machine learning models. Each feature could take value *1* or *0* if the n-gram represented by the feature was or was not generated by the sample respectively. The sequence of memory access operations is a sequence of *Reads* (R) and *Writes* (W). In their work, authors record a first million of memory access operations performed by each executable after it was launched. Afterwards, the sequence of memory access operations is being split into the set of overlapping n-grams of a size 96. Since memory access operations take only two possible values (R and W), the potential feature space of the above-mentioned approach is 2^{96} if a

sequence of memory access operations would be completely random. However, as the same authors mention in their next paper [5], the memory access operations are not random. Thus in [8] their initial feature space is "only" about 6M of features. They had 1204 samples divided into two classes. This can be considered a good sample size based on what was suggested in [11] [15]. However, the amount of features generated under such experimental design makes it impossible to follow "the rule of 10". A straightforward approach in such conditions could be to simply use all the data for training the machine learning model. However, just the storage of a complete dataset from [8] would take more than 6GB of space. Popular machine learning frameworks such as Weka [19] or Scikit-learn [26] are not suited to load and handle so much data. This shows a need for dimensionality reduction. In the works similar to [8] or [5] it is important to keep the original features in order to be able to interpret results. For example, having the results from [8] it might be possible to understand which memory access patterns make malicious behavior distinctive from the benign behavior. Thereby, dimensionality reduction methods such as feature extraction or PCA are not applicable in such cases. On its turn, feature selection can help to select a subset features without hindering their original state.

Feature selection methods can be divided into several categories: filter, wrapper and embedded methods. Filter methods choose features based on a certain quality measure such as Pearsons correlation, Chi-square, mutual information and so on. Wrapper methods choose features based on the classification performance of the target machine learning model trained with the use of those features [33]. Wrapper methods are very computationally intense since for every possible feature subset there is a need to train and test the machine learning model. Embedded methods, as the name states, are embedded in the machine learning algorithms. Algorithms such as Decision Trees [22] perform feature selection simultaneously with model training. However, the computational overhead is higher than one of the filter methods and such algorithms are susceptible to overfitting [9] and are not suitable for high dimensional data [33]. So for the research similar to [8] the most suitable approach for dimensionality reduction will be a filter-based method. In the case of (very) high dimensional data, it is crucial to have a feature selection method with the lowest possible computational overhead. The perfect feature selection method will have a computational complexity of $O(n)$ that is linear to a number of features n. But such a method does not exist, since filter methods are aimed to select features that *represent* classes (and consequently samples) in the best possible way [22]. Thereby, while choosing the feature selection method to work on the high dimensional dataset it is desirable to choose a method with the computational complexity of $O(mn)$ where m is the number of samples in the dataset.

The use of different filter-based feature selection methods are described in various papers. Information Gain [7] [24], Correlation-based feature selection [5] [17] and ReliefF [17] are some of the common feature selection methods. *Information Gain* (IG) ranks features based on entropy in respect to the classes and can be described as "the amount of information, obtained from the attribute A, for determining the class C" [22]. Basically, in order to perform a feature selection based on IG one have to calculate probabilities of an attribute to take certain values and relevant class-

conditional probabilities. This results in a computational complexity around $O(mn)$, where n is the amount of features and m is the amount of samples. *Correlation-based Feature selection method* (CFS) was proposed in [18] and is aimed at selecting the subset of features that have a high correlation to the class but low correlation between each other. By doing so it is possible to find a subset of features with minimal redundancy. The problem with this method, is that it requires to calculate a pairwise correlation matrix between all of the n features and m classes which requires $m((n^2 - n)/2)$ operations. The feature selection search could require an additional $(n^2 - n)/2$ operations in a worst-case scenario. With a potential computational complexity of $O(m((n^2 - n)/2) + (n^2 - n)/2)$ the use of CFS for high dimensional data becomes very problematic. For example, just storing of correlation matrix needed for 6M of features in [8] would require at least 18 TB of space. Thus, in order to apply CFS on high dimensional datasets it might be useful to first reduce a feature space with another, less computationally intense, feature selection method and only after apply the CFS [5]. *ReliefF* ranks features based on their ability to separate close samples from the different classes [22]. In order to perform feature selection with ReliefF, it is first important to calculate a distance matrix between all samples. The resulting computational complexity of the method can be roughly estimated as $O(n((m^2 - m)/2))$ that is almost *m/2* times more than the one of the IG. Having a large n makes the use of ReliefF less favorable than IG.

Based on the assumptions about the computational complexity of the above-mentioned feature selection methods one can make a conclusion, that IG might be one of the best choices when it comes to the high dimensional datasets. The problem is that even the feature selection methods with $O(mn)$ complexity become slow with the large numbers of n. And as we mentioned above, common machine learning packages are not suitable to work with big datasets. Thus, a researcher that needs to perform feature selection on such datasets is forced to develop a custom implementation of feature selection algorithm with regards to the data in interest. In this case, inefficient implementation of the common feature selection algorithm may result in significant use of time and even inability to obtain results (e.g. due to the lack of virtual memory). For example, the Information Gain of a feature is calculated with the following formula:

$$Gain(A) = -\sum_{k} p_k \log p_k + \sum_{j} p_j \sum_{k} p_{k|j} \log p_{k|j}$$

where p_k is the probability of the class k, p_j is the probability of an attribute to take j_{th} value and $p_{k|j}$ is the conditional probability of class k given j_{th} value of an attribute [22]. This shows, that it is necessary to "count" how many times each attribute takes a certain value in total and when a certain class is given. Lets rewrite previously mentioned computational complexity of IG as $O(nT_{qmeaureIG})$ where $T_{qmeaureIG} = f(m)$ is the computational time needed to calculate the quality measure (Information Gain in this case) of a feature. We will need $T_{qmeaureIG}$ later, to show that the IS feature selection method works faster than IG, which is important when working with high dimensional datasets. Thus, it is easy to see that

the inefficient implementation of IG can significantly increase the time needed to obtain the results. As we will later show, it is possible to overcome this problem with a Intersection Subtraction feature selection method.

2.2 Literature overview

In this subsection, we refer to papers where authors addressed the problems related to HDLSS datasets and feature selection on them. In the [12] authors outline both *curses* and *blessings* of high dimensionality. By blessings of dimensionality, they mention the phenomenon of measure concentration and the success of asymptotic methods. While talking about curses of dimensionality authors outline several areas where they can occur: optimization, function approximation and numerical integration. They also stress attention to the fact, that many "classical" statistical methods are based on the assumption, that the amount of features n is less than the amount of samples m, while $m \to \infty$. However, these methods may fail if $n > m$, especially when $n \to \infty$. Other authors in [14] outline the following challenges of high dimensionality: "(i) high dimensionality brings noise accumulation, spurious correlations and incidental homogeneity; (ii) high dimensionality combined with large sample size creates issues such as heavy computational cost and algorithmic instability" [14]. As well as authors of [12] they outline, that traditional statistical methods may fail when used on high dimensional data. The authors of [40] review the performance and limitations of several common classifiers such as Naive Bayes, Linear Discriminant Analysis, Logistic regression, Support Vector Machines and Distance Weighted Discrimination in the case of two-class classification problem on HDLSS datasets. They also say, that if the number of features $n \to \infty$ and both classes are from the same distribution "the probability that these two groups are "perfectly" separable converges to 1" [40]. In simple words, it means, that with a large enough amount of features it should be possible to construct a set of rules (build a classifier) that will perfectly fit (overfit) the training data. This fact outlines the importance of thorough feature selection. It will improve the capability of machine learning algorithms to create models with good *generality* and *interpretability*. The model with good generality is the model that is capable of generalizing over the dataset; such model would not be significantly changed if the number of samples in the dataset is slightly increased/decreased [40]. A model with good interpretability makes the analysis of the model itself easier. The fewer features are involved during the training the easier it is to analyze the obtained model. For example, authors of [5] underline the importance of the fact, that having 29 features instead of 6M or 15M helps in the understanding of the underlying processes. They performed multinomial (10 class) malware classification with the use of features constructed from memory access patterns. Similarly to [8], they used memory access 96-grams as features. Such feature, if found to be important in the classification, can not be directly understood by a human analyst. Thus, in [6] they made an attempt to interpret memory access sequences with more high-level system events (API calls). Such analysis would be

much harder if they had millions of features instead of 29.

Various authors addressed the problem of feature selection on HDLSS datasets more specifically. For example same authors in [36] and [37] present possible improvements to the PCA in HDLSS cases. In [36] they propose a way to estimate singular value decomposition of the cross data matrix. Later, in [37] authors explore the impact of the geometric representation of HDLSS data on a possibility to converge the dataset to an n-dimensional surface. The authors of [13] propose a nonlinear transformation of HDLSS data. They showed, how transformation based on interpoint distances helps to increase final classification accuracy. In the [39] the authors propose a hybrid feature selection method that is based on antlion optimization and grey wolf optimization methods (ALO-GWO). They evaluate the performance of the proposed method on several HDLSS datasets. The authors show that the ALO-GWO feature selection method provides a good balance between the performance of models and the ability to reduce a feature space. The above-mentioned papers addressed the problem of feature selection on HDLSS. However, the number of features in the dataset used in those papers rarely exceeded several tens of thousands (e.g. in [39]). On their turn, authors of [16] during the test of their feature selection method used a dataset with more than 3M of features. In their work, they proposed a feature selection method based on bijective soft sets (BSSReduce). They claim, that the computational complexity of the method is $O(m)$ where m is the number of samples. This might have been a perfect feature selection method for the HDLSS datasets. However, after reviewing the provided algorithms, it looks like their approach relies on the precomputed bijective soft sets that have to contribute to the computational complexity as well. Nevertheless, the results of testing the BSSReduce on the several HDLSS datasets showed, that it is capable of significant dimensionality reduction while keeping a competitive level of the trained models performance. It could be useful to compare BSSReduce with our method, unfortunately, authors of BSSReduce did not provide the source code of their tool. An approach different from the previously mentioned papers is present in the [5]. The authors of the paper did not focus on feature selection. However, they needed to reduce feature space in two HDLSS datasets from 6M and 15M of features. Authors said that "models should be simple enough" [5] to make their analysis easier. In order to reduce a large feature space, they performed feature selection in two steps. On the first step, they used custom implementation of Information Gain feature selection to reduce feature space to 50K and fewer features. On the second step, they took the best 5K feature selected by IG and used them in CFS implementation from Weka. This resulted in 29 features selected by CFS. The models trained with just 29 features performed almost as good as a model trained on 5K and more features. For Naive Bayes and Support Vector Machine algorithms, smaller feature set even allowed to increase the performance of trained models. Such approach has its own limitations. CFS is aimed at selecting features that are not correlated with each other. However, since the first feature selection step utilizes IG, there is no guarantee that features passed to the CFS does not have a strong mutual correlation. But as we mentioned above, running CFS on the HDLSS dataset with millions of features requires enormous computational resources and sometimes impossible.

3 Intersection Subtraction selection method

In this section, we describe the IS feature selection method and evaluate its strengths and weaknesses.

3.1 The context

Before describing the Intersection Subtraction feature selection method we need to describe a context under which its use becomes meaningful. This method was developed during the research described in [8]. The task was to detect malware based on the memory access traces. To do this, malicious and benign executables were launched together with custom-built Intel Pin [20] tool. The raw data consisted of the first 1M of memory access operations performed by each executable. The sequences contained *W* for each write operation and *R* for each read operation performed by an executable. These sequences were later divided into a set of overlapping n-grams of various sizes. For example, a sequence *[WWRWRR]* of a length 6 can be divided into the set of 4-grams in the following way: *[WWRW,WRWR,RWRR]*. The n-grams were directly used as features for machine learning models training. Each feature got value *1* if the corresponding n-gram was generated by the sample regardless of the number of times it was encountered in the trace of a certain sample. In other cases, the feature got value *0*. As the goal of the [8] was to be able to detect malware, it is possible to state, that features that obtain *1* (are present within a certain class) pose greater interest. Such approach allows to state, that *presence* of certain memory access n-grams is the sign of malicious behavior. The dataset from [8] was nearly balanced and samples were divided into two classes. So the context of the use of the proposed feature selection method is the following: two-class classification problem on a balanced dataset with binary features.

3.2 Feature selection algorithm

The feature space in [8] was around 6M of unique memory access n-grams of a size 96. By the time of writing, authors were not able to implement any common feature selection method (for example IG) to operate on such dataset. Thus, they implemented the following feature selection method. It includes the following steps:

1. Construct two vectors of features for each class. The feature is included in the vector of the class if the corresponding memory access n-gram was generated by a sample from this class.
2. Having two vectors constructed, remove from them features that are present in both vectors. Having this done we obtain two vectors of class-unique features.

In other words, we *subtracted an intersection* of two feature sets from both of them.
3. Decide on the size of the final feature set k.
4. From each of the class-unique features vectors select *k/2* features with the highest class-wise frequency. A class-wise frequency is the proportion of samples within the class that generate a corresponding memory access n-gram.
5. Use the k selected features to construct the final dataset with reduced dimensionality.

The resulting dataset is later used to build machine learning models. The operation performed in Step 2 is quite similar to the symmetric difference of two sets. However, we prefer to say that we subtract intersection from both sets, as we need those sets to be separated until the last step. It is also worth mentioning, that having an intersection of two feature sets allows to explore features that fell into it. It might be useful for additional analysis of the results [8].

3.3 Computational complexity

Lets discuss the potential computational complexity of Intersection Subtraction (IS) feature selection. As data is already labeled (samples divided into two classes) the feature vectors from the *Step 1* are ready from the beginning. *Step 2* requires finding an intersection of two sets. Imagine we have two sets A and B with cardinality of a and b respectively. In order to find the intersection of A and B we need to compare all elements of set A with all elements of set B. Such operation will have a computational complexity of $O(ab)$. Let's denote the intersection of A and B as $C = A \cap B$ with cardinality c. Subtracting the elements of C from A and B, similarly to the previous operation, will have the computational complexity of $O(ac + bc)$. The resulting computational complexity of $O(ab + ac + bc)$ may look quite high already, since both a and b are large in case of HDLSS datasets. However, the real implementation of IS feature selection with the use of Python programming language shows, that execution of the *Step 2* does not take significant time (see Section 4). First of all, according to [28], subtraction *A-C* (set difference) will have computational complexity of $O(a)$. So we can already rewrite previously mentioned computational complexity of Step 2 with $O(ab + a + b)$. Moreover, if we are not interested in the intersection C itself, we can utilize two operations *A-B* and *B-A* in order to obtain sets of class-unique features. Complexity of such approach will be $O(a+b)$. The *Step 4* requires the calculation of class-wise frequencies of the features. In our particular case, when features are binary, we only need to count how many samples from each class has value *1* of a certain feature. The Step 4 will then have $O((a-c)m+(b-c)m)$ computational complexity. Here, m is the number of samples in the dataset, *a-c* is the amount of class-unique features from set A and *b-c* - from set B. It is also worth mentioning, that Step 4 can be optimized. Let's assume that the dataset is perfectly balanced, so we have two classes with *m/2* samples. Since our IS feature selection is aimed on finding class-unique features, we can only search for *1s* among *a-c* and

b-c features of *m/2* samples of each class. So the Step 4 can be optimized to have a complexity of $O((a-c)m/2 + (b-c)m/2)$. Lets now try to assess the overall computational complexity of the IS feature selection. Let us have the initial amount of features $a+b=n$ and m samples. The amount of features from intersection c is normally smaller than both a and b (here we assume, that $A \not\subset B$ and $B \not\subset A$). Having this we can conclude, that the complexity of *Step 2* $O(ab + a + b)$ after substitution will be smaller than $O(n^2)$ for all $a > 1$. On its turn, the complexity of *Step 4* $O((a-c)m/2 + (b-c)m/2)$ should be smaller than $O(mn)$. The resulting complexity of $O(ab + a + b + (a-c)m/2 + (b-c)m/2)$ should be smaller than $O(n^2 + mn)$. The feature selection method where the upper boundary of computational complexity is described with n^2 is not what we outlined in Section 2 as a good feature selection method for HDLSS dataset. Lets now make a substitution similar to the one we made in Section 2. First, lets substitute m with $T_{qmeaureIS} = g(m)$ which is the time needed to calculate class-wise frequency of a feature. Second, the time T_{in} needed to find whether a certain feature from one set is present *in* another set (to find an intersection, or to subtract these features from the set) is relatively small. Thus, the updated computational complexity of IS feature selection will be smaller than $O((nT_{in})^2 + nT_{qmeaureIS})$ which can be smaller than $O(nT_{qmeaureIG})$ of IG. We will prove this in Section 4.

3.4 Theoretical assessment

In this subsection, we discuss potential outcomes of the IS feature selection. As we already mentioned, IS feature selection is potentially faster than a more common IG feature selection. This makes IS attractive for the high dimensional datasets. However, speed comes with a price. Let's look at the potential disadvantages of IS feature selection. As we described at the beginning of this section, the use of this method makes more sense when we are interested in finding features the *presence* of which poses particular interest. However, it might happen, that in the dataset will be no class-unique features. In other words, it will be impossible to say, that if a certain feature of a sample takes value 1, then this sample belongs to a certain class. In such case, it will be impossible to find an intersection of two feature sets. The other problem is potential information loss due to intersection removal. Imagine we have a dataset that is represented in the Table 1. It has 4 features and 4 samples labeled into two classes C1 and C2. IS feature selection will remove features f1 and f3 since they obtain value 1 (are present) in both classes. The remaining features f2 and f4 will not allow us to generate a rule that will be able to distinguish between samples s2 and s4. This example is quite small, but on the larger dataset removing a feature that takes value *1* in e.g. all samples of one class and only in one sample of another class can lead to the inability of building a model with good performance. Such feature would be most likely selected by IG feature selection. The last disadvantage of the IS feature selection is potentially poor performance on the multinomial datasets. If we increase the number of classes we will end up in the situation of growing

Table 1 Sample dataset 1

	f1	f2	f3	f4	
s1	1	1	1	0	C1
s2	1	0	0	0	C1
s3	1	0	1	1	C2
s4	0	0	1	0	C2

intersection size. In such case, the IS will remove more features from the feature space resulting in increased information loss. We begin with the description of our dataset and experimental environment. Later, we explain the basics of memory access operations and explain the way we record and process the data.

4 Experimental evaluation

In this section we describe experimental evaluation of the IS feature selection method. We show how IS feature selection can be applied for malware detection. During experimental evaluation we compare performance of features selected by IS and IG. On the Figure 1 we depict general data-flow of our experiments. We start by recording memory access operations produced by benign and malicious executables. After, we split sequences of memory access operations into n-grams. Then we apply feature selection methods to select best features (n-grams). In the end, we use these features to train machine learning models and compare performance of the models trained with a use of features selected by different feature selection methods. Before presenting the results achieved by machine learning models, we show the experimental time complexity of the IS and IG feature selection methods. We also check how similar the feature sets selected by different methods are.

We now proceed with the description of our dataset, experimental environment and the way we collect and process the data.

4.1 Dataset

In this work we use dataset similar to the one used in [7]. It consists of 2098 benign and 2005 malicious Windows executables. Malicious executables were downloaded as part of *VirusShare_00360* pack available at VirusShare [35]. Malicious samples belong to the following malware families: Fareit, Occamy, Emotet, VBInject, Ursnif, Prepscram, CeeInject, Tiggre, Skeeyah, GandCrab. According to the VirusTotal [32] reports, our samples were first seen (first submission date) between March 2018 and March 2019. Benign executables are the real software downloaded from Portable Apps [27] in September 2019.

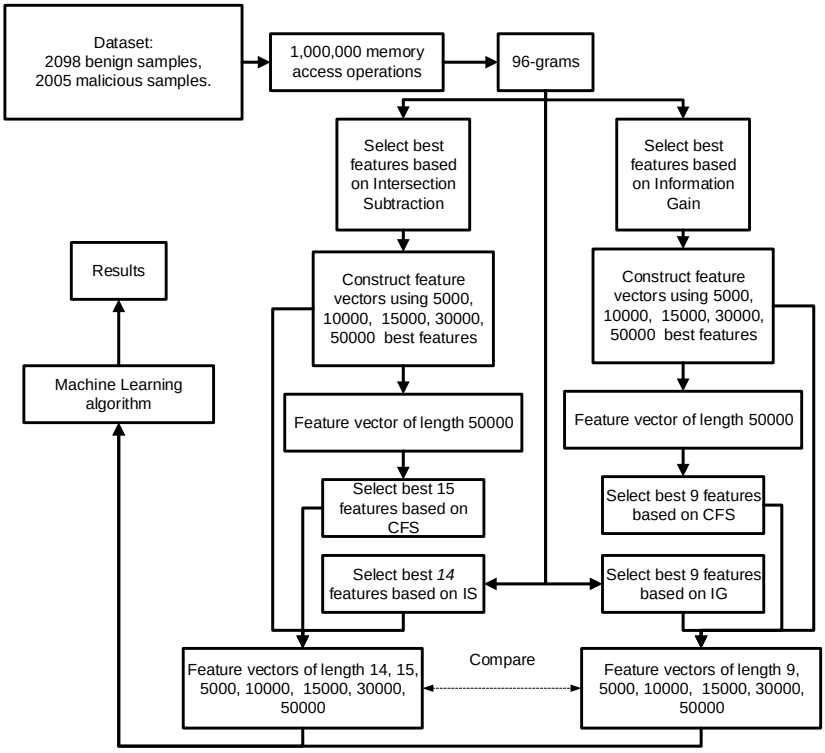

Fig. 1 The flow of data collection and feature selection

4.2 Experimental environment

In order to perform dynamic malware analysis, we need to avoid the influence of any environmental changes, so that all executables are launched in similar conditions. To achieve this we used an isolated Virtual Box virtual machine (VM) with Windows 10 guest operating system. VMs were launched on the Virtual Dedicated Server (VDS) with 4-cores Intel Xeon CPU E5-2630 CPU running at 2.4GHz and 32GB of RAM with Ubuntu 18.04 as a main operating system.

4.3 Memory access operations

The executables used on Windows operating systems are compiled into the files in PE32 format. Files of PE32 format contain *header* and *sections*. The header contains the metadata that is used by operating system in order to properly load

an executable into memory and prepare all the necessary resources. The sections contain information about imported and exported functions, resources, data and the executable code. The executable code is stored in the binary form which can be represented as *opcodes*. Opcodes (or assembly commands) are the basic instructions that are executed by the CPU. Execution of some instructions will not require memory access. Foe example execution of *MOV EAX,EBX* opcode will not result in memory access, since data is being moved between registers in the CPU. At the same time, *MOV EDI, DWORD PTR [ebp-0x20]* will generate a *Read (R)* memory access, since the data has to be read from the memory. On its turn, the *ADD DWORD PTR [EAX],ECX* will require *Reading (R)* the value from the memory location addressed by *[EAX]* and then *Writing (W)* the result of the addition to the memory. The sequences of opcodes were previously proven to be a source of effective features for malware detection [10] [34] [30]. When the sequence of opcodes is executed it generates a sequence of memory access operations. Two previous statements allow for memory access sequences to be a potential source of features for malware detection [8]. Under our experimental design we use only the type of memory access operation: *R* for Read and *W* for Write. We do not use the value that is transferred to or from the memory as well as the address of the memory region in use.

4.4 Data collection

Each malware sample was launched on the clean snapshot of VM. During the execution of each sample, we recorded the first million of memory access operations produced after the launch. This was done with the help of a custom-built Intel Pin [20] tool that was launched together with the sample inside the VM. The VM had all built-in anti-virus features disabled to make malware run properly and also because they kept interrupting the work of Intel Pin. The automation of VM and data collection were performed with the help of Python 3.7 scripts.

The memory access traces were first stored in the separate files. After, they were split into the sequence of overlapping n-grams of the size 96 (96-grams). We choose n-gram size (as well as the amount of recorded memory accesses) based on the conclusions of their effectiveness drawn in [8]. The n-grams of memory access operations for each sample are then stored in the MySQL table. This table took 28.5 GB of storage.

4.5 Feature selection and machine learning algorithms

We implemented IS feature selection algorithm with Python. The custom implementation of IG feature selection algorithm was similar to one in [7]. That implementation allows to run feature selection in multiple threads, which significantly speeds up the process. We found, that samples produced more than 5.5M of unique n-grams (fea-

tures). Benign samples generated more than 4.5M of features, while malicious - more than 1M of features. When performing IS feature selection we found, that benign and malicious samples shared almost 600K common features. Subtraction of those features resulted in almost 4M and 430K of class-unique benign and malicious features respectively. According to the algorithm from Section 3 we selected 50,30,15,10 and 5 thousands of features. We selected a similar amount of features with the IG feature selection algorithm as well. Similarly to [5], [6] and [7] we wanted to reduce feature space even more, so that our models are simple enough for future human analysis. Thus, we used CFS feature selection method from Weka [19] to select the most relevant and least redundant features from 50K features selected by IS and IG. As the result we obtained 15 features from IS-based 50K feature set, and 9 features from IG-based 50K feature set. As CFS appends features to the feature set until the increase of its merit is no longer possible, it is impossible to control the final amount of selected features unless the GreedyStepwise search is applied. However, such search never finishes its work when applied to the larger feature sets in our experimental environment. We wanted to compare the performance of IS and IG with the CFS as well. So we tried to select the same number of features with IG and IS. However, CFS selected 15 features. And as the IS have to select equal amount of features from each class (Section 3) we decided to select 14 features with IS (7 from each class).

The selected features were later used to build machine learning models. The data that is actually fed into mechine learning algorithms is basically a *bitmap of presence* [8]: if a certain sample (row) generates a certain feature (column), then this feature takes value *1* for this sample. In the opposite case the feature takes value *0*. We used the following machine learning algorithms from Weka: k-Nearest Neighbors (kNN), RandomForest (RF), Decision Trees (J48), Support Vector Machines (SVM) and Naive Bayes (NB) with the default Weka [19] parameters. We assessed the quality of the models with 5-fold cross validation [22]. Accuracy (ACC) as the amount of correctly classified samples and F1-measure (F1M) that takes into account precision and recall were chosen as evaluation metrics. Further in this section, we present the classification performance of the machine learning models.

4.6 Time complexity

One of the reasons to use IS feature selection is that it is relatively faster than the other common methods. In this subsection, we provide time taken by IS and IG methods to select 50K of features from the initial 5.5M distinct features. It took 302 seconds (~5 minutes) for IS to select 50K features. In contrast, the IG used 18,560 seconds (~5.15 hours) to select 50K features when running in one thread. While being launched in 16 threads, IG used 1168 seconds (~20 minutes) to select 50K features. Further increase in the number of threads does not make sense, since this is the maximum amount of threads available at our VDS. As we can see, single-threaded IS works 3.8 times faster than IG ran with 16 threads and 61.5 times faster than IG ran with

one thread. To find an intersection of benign and malicious feature sets the IS used 1.18 seconds as the average of 1000 runs. It has used an additional 0.7 seconds to subtract intersection from both feature vectors. The actual implementation of our feature selection algorithms did not load the entire dataset at the same time. Thus, it is impossible to directly measure the time needed to calculate the quality measure of a single feature, since it is calculated in iterations. But indirect assessment (we divide overall time by the total amount of features to go through) showed, that IS needed around $5.5 \cdot 10^{-5} s$ to assess a single feature, and IG needed $2.12 \cdot 10^{-4} s$ and $3.4 \cdot 10^{-3} s$ to assess a single feature with 16 and 1 thread respectively. It is important to mention, that the times provided are relevant to our data structure and the way we store our data. For instance, the fact that we stored memory access n-grams for each sample in a separate cell of the database table could affect the time needed to perform feature selection.

4.7 Analysis of selected feature sets

Here we analyze how different are the feature sets selected by IS and IG. In the Table 2 the Feature amount column shows the size of the feature set for IS and IG methods; the Common features column shows the number of similar features selected by IG and IS for the corresponding feature set size; the Difference ratio column shows the ratio of the distinct features and is calculated as *(Feature amount - Common features)/Feature amount*. As we can see, most of the features selected by

Table 2 Difference between feature sets selected by IS and IG.

Feature amount	Common features	Difference ratio
50K	994	0.98
30K	994	0.97
15K	979	0.93
10K	955	0.9
5K	812	0.84
IG/IS 9/14	0	1

the IS method are different from those selected by IG. It complies with the theoretical assessment of IS (see Section 3), where we explained that IS may discard features with potentially high information gain only because they get value *1* in both classes. As we mentioned before, we used CFS feature selection on the feature sets of the size 50K. It is worth mentioning that CFS selected completely different features when working with 50K feature sets selected by IS or IG. When using IG and IS to select the same amount of features as selected by CFS we also obtained completely different feature sets.

4.8 Classification performance

In this subsection, we present the classification performance achieved by the machine learning algorithms. Tables 3 and 4 contain evaluation metrics of machine learning models trained with the feature sets of a different length selected by different feature selection algorithms. There, *FSL* stands for feature set length, *ACC* stands for accuracy and F1M stands for F1-measure. As we can see, both feature vectors allowed to achieve a quite high classification accuracy. The best performing RF model that used 10K features selected by IG managed to classify 99.9% of the samples correctly. On its turn, features selected by IS allowed to build kNN and RF models with an accuracy of 99.8%. As we can see, in most cases models built with the use of features selected by IS have slightly lower classification performance. However, the difference in accuracy or F1-measure between IS and IG features is most of the time less than 1%. Thus, it is hard to conclude whether the features selected by IG is significantly better than those selected by IS. There is one exception for NB models built with the use of 50K features. As it is possible to see, the NB model trained with 50K features selected by IG has significantly lower accuracy and F1-measure than the one trained with 50K features selected by IS. This difference might be explained by the nature of features selected by IS and the limitations of the NB method. While building the model, Naive Bayes assumes that features are independent. However, Information Gain feature selection potentially selects a lot of mutually correlated features. The IS does not take into account the mutual correlation between features as well. However, there should be less correlated features selected by IS, since one half of the features will not have *1s* in one of the classes and vice versa. These properties of Naive Bayes were studied in [29]. Even though CFS selected completely different features in IS and IG cases, the models built with those features showed a quite similar classification performance. We will discuss this in Section 5. When we used IS and IG to select the number of features similar to CFS we found, that models built with these features perform slightly worse if compared to the models built with features selected by CFS. This finding can be explained by the natures of CFS and IS algorithms. The IS will select features with higher class-wise frequency. However, such features might correlate with each other. Thus, these features might have a strong correlation with each other bringing redundant information to the model. In contrast, CFS will try to select a feature set that has as little redundant information as possible. Looking once again in the Tables 3 and 4 we can conclude, that both feature selection methods performed quite good under our experimental setup while selecting feature sets that are very different to each other.

Important notice. The results from the Table 3 is similar to part of the results provided in [7]. This happened because our papers share the same dataset. Also the data collection processes have only minor differences: in this paper we recorded the first million of memory access operations, while methodology of [7] is to record the first million of memory access operations unless a certain stopping criteria is met.

Table 3 Classification performance with a use of features selected by IG

		kNN		RF		J48		SVM		NB	
Method	FSL	ACC	F1M	ACC	F1M	ACC	F1M	ACC	F1M	ACC	F1M
InfoGain	50K	0.996	0.996	0.996	0.996	0.997	0.997	0.983	0.983	0.693	0.671
	30K	0.996	0.996	0.997	0.997	0.998	0.998	0.986	0.986	0.983	0.983
	15K	0.996	0.996	0.998	0.998	0.998	0.998	0.991	0.990	0.983	0.983
	10K	0.998	0.998	**0.999**	**0.999**	0.998	0.998	0.992	0.991	0.983	0.983
	5K	0.995	0.995	0.997	0.997	0.997	0.997	0.988	0.988	0.983	0.983
	9	0.988	0.988	0.988	0.988	0.988	0.988	0.988	0.988	0.988	0.988
CFS	9	0.997	0.997	0.997	0.997	0.996	0.996	0.997	0.997	0.988	0.988

Table 4 Classification performance with a use of features selected by IS

		kNN		RF		J48		SVM		NB	
Method	FSL	ACC	F1M	ACC	F1M	ACC	F1M	ACC	F1M	ACC	F1M
IS	50K	0.991	0.991	0.997	0.997	0.997	0.997	0.983	0.983	0.982	0.982
	30K	0.996	0.996	0.997	0.997	0.997	0.997	0.983	0.983	0.985	0.985
	15K	**0.998**	**0.998**	**0.998**	**0.998**	0.997	0.997	0.984	0.984	0.983	0.983
	10K	0.998	0.998	0.997	0.997	0.997	0.997	0.985	0.985	0.983	0.983
	5K	**0.998**	**0.998**	**0.998**	**0.998**	0.997	0.997	0.985	0.985	0.983	0.983
	14(7+7)	0.983	0.983	0.983	0.983	0.983	0.983	0.983	0.983	0.983	0.983
CFS	15	0.997	0.997	0.997	0.997	0.997	0.997	0.997	0.997	0.983	0.983

5 Discussion and Future work

In this section we discuss our findings and limitations that should be applied to the possible conclusions made based on the presented results. As we were able to see, IS feature selection works faster than IG. The main reason to this is the fact that the selection of features based on its class-wise frequency requires less computations. However, it is important to understand, that all measurements of time complexity presented in this paper are specific to our conditions (available computational resource, structure of the data, implementation of feature selection algorithms) and might differ in other conditions. The theoretical assessment of the IS feature selection method predicted, that features selected by IS might bring less information about samples and classes than those selected by IG. But the experimental evaluation showed only marginal difference in classification performance. Under our experimental setup, only the amount of features selected by CFS could be considered as a proof of our theoretical assessment. The CFS selected more features from IS-selected feature set to gain similar merit (what resulted in similar classification performance). As we mentioned before, CFS adds features to the feature set until its merit stops growing. These facts show, that features selected by IS possess less information. Thus, on the small feature sets, we need more features selected by IS than those selected by IG. As we compared classification performance of machine learning methods we found, that under certain conditions NB might perform better when using IS-selected features. This fact can be explored more thoroughly in the future work. The method was tested on a nearly balanced dataset, and we selected the equal amount of features to represent both classes. The use of other approach in the selection of the desired

amount of features or applicability on the imbalanced datasets is left for the future work.

The IS feature selection method is quite simple in implementation. However, as we discussed in Section 3, its applicability limited to the cases where we are interested in the fact of presence of a certain feature in the class. Thus, when features are not binary or discreet, the applicability of IS feature selection is questionable. It is possible, however, to binarize continuous variables [22], but this a separate topic and it is out of scope of this paper. There is also a number of possible improvements and modifications that can be applied to the IS feature selection method in the future. For example, we can decrease the time complexity of IS in the following way. When we calculate class-wise frequencies of features we might limit the search space by the samples that *produce* this feature. Rough estimation suggest, that it may halve the time needed to perform IS feature selection. Another modification that can be implemented in IS feature selection is introduction of the degree of membership to the intersection. For example, a certain feature f might occur in both classes *C1* and *C2*. These classes have m_{C1} and m_{C2} samples respectively. The feature f is present in m^f_{C1} samples of a class *C1* and m^f_{C2} samples of class *C2*. For example, we may exclude feature from the intersection if:

$$\frac{max(\frac{m^f_{C1}}{m_{C1}}, \frac{m^f_{C2}}{m_{C2}})}{min(\frac{m^f_{C1}}{m_{C1}}, \frac{m^f_{C2}}{m_{C2}})} > \epsilon$$

Basically, we keep a feature if it represents ϵ times bigger fraction of samples of one class than fraction of samples of the other class. Such approach may decrease an information loss, but will contribute to the increase of computational complexity of IS feature selection method. And thus will make IS less attractive feature selection method.

It is also important to outline the following observation. IS and IG selected quite different feature sets. Moreover, CFS selected completely different features from those preselected by IS and IG. Nevertheless, classification performance of the machine learning models appeared to be very similar when using different feature sets. This raises the following question: do the mentioned feature selection methods always select the best feature set or do they find *one* of the several similarly good feature sets? This question is left open for the future studies.

6 Conclusions

In this paper, we studied the performance of Intersection Subtraction feature selection on malware detection problem. We showed, that with the use of IS feature selection on HDLSS dataset it is possible to correctly classify more than 99% of the benign and malicious samples. The main contribution of this paper is the direct comparison of IS and IG feature selection methods under the same conditions. We found, that most of

the features selected by IS and IG are different. The classification performance of the machine learning models trained with the use of quite different feature sets appeared to be very similar. Even though the models trained with IG-selected features showed marginally better performance, the single-thread implementation of the IS feature selection method worked 3.8 times faster than the 16-threads implementation of IG. This makes Intersection Subtraction feature selection attractive when it comes to the analysis of HDLSS datasets. The IS feature selection may help when it is not known yet whether the data is useful for the classification task at all. The number of features might so big, that it is pointless to spend time running more common (also slower) feature selection methods. Thus, with certain above-mentioned limitations, the IS feature selection may be successfully applied to HDLSS datasets.

References

1. Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. Information security governance: the art of detecting hidden malware. In *IT security governance innovations: theory and research*, pages 293–315. IGI Global, 2013.
2. Manoun Alazab, Robert Layton, Sitalakshmi Venkataraman, and Paul Watters. Malware detection based on structural and behavioural features of api calls. 2010.
3. AVTEST. The independent IT-Security Institute. Malware. `https://www.av-test.org/en/statistics/malware/`, 2020.
4. Ahmad Azab, Mamoun Alazab, and Mahdi Aiash. Machine learning based botnet identification traffic. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1788–1794. IEEE, 2016.
5. Sergii Banin and Geir Olav Dyrkolbotn. Multinomial malware classification via low-level features. *Digital Investigation*, 26:S107–S117, 2018.
6. Sergii Banin and Geir Olav Dyrkolbotn. Correlating high-and low-level features. In *International Workshop on Security*, pages 149–167. Springer, 2019.
7. Sergii Banin and Geir Olav Dyrkolbotn. Detection of running malware before it becomes malicious. page To be published, 2020.
8. Sergii Banin, Andrii Shalaginov, and Katrin Franke. Memory access patterns for malware detection. *Norsk informasjonssikkerhetskonferanse (NISK)*, pages 96–107, 2016.
9. Max Bramer. *Principles of data mining*, volume 180. Springer, 2007.
10. Domhnall Carlin, Philip O'Kane, and Sakir Sezer. Dynamic analysis of malware using run-time opcodes. In *Data analytics and decision support for cybersecurity*, pages 99–125. Springer, 2017.
11. Kevin K Dobbin and Richard M Simon. Sample size planning for developing classifiers using high-dimensional dna microarray data. *Biostatistics*, 8(1):101–117, 2007.
12. David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.
13. Subhajit Dutta and Anil K Ghosh. On some transformations of high dimension, low sample size data for nearest neighbor classification. *Machine Learning*, 102(1):57–83, 2016.
14. Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *National science review*, 1(2):293–314, 2014.
15. Rosa L Figueroa, Qing Zeng-Treitler, Sasikiran Kandula, and Long H Ngo. Predicting sample size required for classification performance. *BMC medical informatics and decision making*, 12(1):8, 2012.
16. Ke Gong, Yong Wang, Maozeng Xu, and Zhi Xiao. Bssreduce an o (u) incremental feature selection approach for large-scale and high-dimensional data. *IEEE Transactions on Fuzzy Systems*, 26(6):3356–3367, 2018.

17. Lars Strande Grini, Andrii Shalaginov, and Katrin Franke. Study of soft computing methods for large-scale multinomial malware types and families detection. In *Recent Developments and the New Direction in Soft-Computing Foundations and Applications*, pages 337–350. Springer, 2018.
18. M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.
19. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.
20. IntelPin. A dynamic binary instrumentation tool, 2020.
21. Khaled N Khasawneh, Meltem Ozsoy, Caleb Donovick, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Ensemble learning for low-level hardware-supported malware detection. In *Research in Attacks, Intrusions, and Defenses*, pages 3–25. Springer, 2015.
22. Igor Kononenko and Matjaž Kukar. *Machine learning and data mining: introduction to principles and algorithms*. Horwood Publishing, 2007.
23. Malay Haldar. How much training data do you need? `https://medium.com/@malay.haldar/how-much-training-data-do-you-need-da8ec091e956`, 2015.
24. Olga Ogorodnyk, Ole Vidar Lyngstad, Mats Larsen, Kesheng Wang, and Kristian Martinsen. Application of machine learning methods for prediction of parts quality in thermoplastics injection molding. In *International Workshop of Advanced Manufacturing and Automation*, pages 237–244. Springer, 2018.
25. Meltem Ozsoy, Khaled N Khasawneh, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Hardware-based malware detection using low-level architectural features. *IEEE Transactions on Computers*, 65(11):3332–3344, 2016.
26. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
27. PortableApps.com. Portableapps.com. `https://portableapps.com/apps`, 2020.
28. Python.org. Time complexity. `https://wiki.python.org/moin/TimeComplexity`, 2020.
29. Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 616–623, 2003.
30. Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231:64–82, 2013.
31. Andrii Shalaginov, Sergii Banin, Ali Dehghantanha, and Katrin Franke. Machine learning aided static malware analysis: A survey and tutorial. In *Cyber Threat Intelligence*, pages 7–45. Springer, 2018.
32. Virus Total. Virustotal-free online virus, malware and url scanner. *Online: https://www.virustotal. com/en*, 2012.
33. B Venkatesh and J Anuradha. A review of feature selection and its methods. *Cybernetics and Information Technologies*, 19(1):3–26, 2019.
34. P Vinod, Vijay Laxmi, and Manoj Singh Gaur. Reform: Relevant features for malware analysis. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, pages 738–744. IEEE, 2012.
35. VirusShare. Virusshare.com. `http://virusshare.com/`. accessed: 09.03.2020.
36. Kazuyoshi Yata and Makoto Aoshima. Effective pca for high-dimension, low-sample-size data with singular value decomposition of cross data matrix. *Journal of multivariate analysis*, 101(9):2060–2077, 2010.
37. Kazuyoshi Yata and Makoto Aoshima. Effective pca for high-dimension, low-sample-size data with noise reduction via geometric representations. *Journal of multivariate analysis*, 105(1):193–215, 2012.
38. Çağatay Yücel and Ahmet Koltuksuz. Imaging and evaluating the memory access for malware. *Forensic Science International: Digital Investigation*, 32:200903, 2020.

39. Hossam M Zawbaa, Eid Emary, Crina Grosan, and Vaclav Snasel. Large-dimensionality small-instance set feature selection: a hybrid bio-inspired heuristic approach. *Swarm and Evolutionary Computation*, 42:29–42, 2018.
40. Lingsong Zhang and Xihong Lin. Some considerations of classification for high dimension low-sample size data. *Statistical methods in medical research*, 22(5):537–550, 2013.