Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection*

Raja Khurram Shahzad† and Niklas Lavesson
School of Computing
Blekinge Institute of Technology
SE-37179, Karlskrona, Sweden
{rks, niklas.lavesson}@bth.se

Abstract

Malicious software (malware) represents a threat to the security and the privacy of computer users. Traditional signature-based and heuristic-based methods are inadequate for detecting some forms of malware. This paper presents a malware detection method based on supervised learning. The main contributions of the paper are two ensemble learning algorithms, two pre-processing techniques, and an empirical evaluation of the proposed algorithms. Sequences of operational codes are extracted as features from malware and benign files. These sequences are used to create three different data sets with different configurations. A set of learning algorithms is evaluated on the data sets. The predictions from the learning algorithms are combined by an ensemble algorithm. The predicted outcome of the ensemble algorithm is decided on the basis of voting. The experimental results show that the veto approach can accurately detect both novel and known malware instances with the higher recall in comparison to majority voting, however, the precision of the veto voting is lower than the majority voting. The veto voting is further extended as trust-based veto voting. A comparison of the majority voting, the veto voting, and the trust-based veto voting is performed. The experimental results indicate the suitability of each voting scheme for detecting a particular class of software. The experimental results for the composite F1-measure indicate that the majority voting is slightly better than the trusted veto voting while the trusted veto is significantly better than the veto classifier.

Keywords: Malware detection, scareware, veto voting, feature extraction, classification, majority voting, ensemble, trust, malicious software

1 Introduction

Malicious software (malware) is a common computer threat and is usually addressed through the static and the dynamic detection techniques. Anti-malware tools are only able to detect known malware instances and the success rate is circa 30% [2] in the wild. In an effort, to extend both the static and dynamic approaches, some researchers apply machine learning (ML) algorithms to generate classifiers, which show promising results both in detecting the known and novel malware. To increase the detection accuracy, the output (prediction) of different classifiers is combined (based on different parameters) to form an ensemble [3]. Ensembles can be data dependent such as multiple algorithm trained on the same data set, or independent from the data, i.e., using statistical measures [4]. The prediction of each participating classifier in the ensemble may be considered as a vote for a particular class, i.e., benign class or malware class. The ensemble's outcome is generally derived on the basis of different voting strategies. Different voting strategies may give different results depending upon different factors such as families

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 4, number: 1, pp. 98-117

*This paper is an extended version of the work originally presented at the 7th International Conference on Availability, Reliability and Security (ARES'12), Prague, Czech Republic, August 2012 [1].

†Corresponding author: Tel: +46-455-385897

of algorithms used. Among different voting strategies, the majority voting is generally used for different problems. The majority voting is considered as a simplest and effective scheme [5]. The majority voting scheme follows democratic rules, i.e., the class with highest number of votes is the outcome. Majority vote does not assume prior knowledge about the problem in hand or classifiers used and may not require training [5]. The majority voting scheme has different limitations such as a subset of classifiers (majority in number) may agree on the misclassification of an instance by a chance. An alternative voting scheme is the veto voting, i.e., one single classifier vetoes the decision of other classifiers. The veto voting scheme is used in the fault diagnosis, the author identification and the malware detection [6, 1]. For the malware detection a ML-based detection model is proposed in which the inductive biases of different algorithms are combined, and the final prediction is given on the basis of veto voting, i.e., if an algorithm predicts the instance as a malware, this prediction may veto all the other predictions and the outcome is malware [1]. The veto voting may also affects the performance of the ensemble as outcome may depend on one single algorithm. However, the veto voting can achieve higher classification accuracy on the assumption that the data set contains abundant instances of a particular class (favored by veto) [7]. Thus, it is worth to investigate which voting scheme, i.e., majority voting or veto voting is suitable for the malware detection. After the comparison of the majority voting and the veto voting, the veto voting is extended from a simple veto voting to the trust-based veto voting. The trust-based veto voting considers the trust of each algorithm to determine whether an algorithm or set of algorithms can veto the decision. The trust-based veto voting is also applied on the same data set and results are compared with the majority voting and the veto voting. The majority voting is suitable for detecting benign applications and lacks the accuracy in the detection of malware in comparison to the veto voting. Similarly, the veto voting is suitable for detecting the malware and is less accurate for the detection of benign. The experimental results indicate that the proposed trust-based veto voting algorithm may be used to overcome the deficiencies of both the majority voting and the veto voting up to some extent. The trust-based veto voting accurately detect both novel and known instances of malware better than the majority voting and accurately predicts about benign instances better than the veto voting. The experimental results also indicate that if composite measure is taken into account, majority voting is slightly better than the trust-based veto voting.

1.1 Aim and Scope

The aim is to evaluate the malware detection methods that combine the output of a set of classifiers and provides a classification on the basis of the majority voting or the veto voting. A malware detection application is developed, which implements the majority voting, the veto voting and the trust-based veto voting. The prediction results from all voting strategies are compared with each other. To achieve better predictive results, the quality of information (i.e., the information, which provides valuable input for classification) derived from the data in the pre-processing stage is very important. Therefore, two pre-processing techniques are also proposed and investigated.

1.2 Contribution

The contributions are: first, a malware detection model is proposed and implemented, which combines the inductive biases of different algorithms and uses contrary voting strategies for the prediction. Second, an extension to a particular voting strategy, i.e., the veto voting is proposed and implemented. Third, the empirical results of different voting strategies are analyzed. Fourth, two pre-processing techniques are proposed to extract features from executable files. These pre-processing techniques can be used to extract both hexadecimal based features or assembly instruction based features of different sizes.

1.3 Outline

The remainder of the article is organized as follows: Section 2 discusses the background, terminology and related work. Section 3 presents the veto-based classification by discussing it's architecture. Section 4 discusses the pre-processing techniques. Section 5 describes the experimental procedure for the first experiment (i.e., Experiment I) and compares the results of the majority voting and the veto voting. Section 6 describes the trust-based veto voting, presents the second experiment (i.e., Experiment II), and analyzes the experimental results. Finally, Section 7 concludes the work and describes future directions.

2 Background

One of the challenges faced by computer users is to keep the information and communication away from unwanted parties who exploit vulnerabilities present in the operating system (OS) or third party software to jeopardize the communication and access the information. A popular way to exploit vulnerabilities remotely is by using a malware [8]. Traditionally, the malware detection is conducted either by using the static analysis, i.e., by matching specific patterns called signatures or on the basis of a rule set, or the dynamic analysis, i.e., changes occurring in the system due to the execution of a specific file. The main deficiency of these techniques is that they fail to detect the zero-day attacks.

The use of ML has been investigated in fields such as natural language processing, medical diagnosis, and malware detection. ML can be divided into two broad categories. In supervised learning, algorithms are used to generate a classifier or a regression function using the labeled data. To achieve the better predictive performance, a finite set of classifiers can be combined as an ensemble. The prediction of most ensembles is based on the majority voting [3]. If the data is incompletely labeled, unsupervised learning is used. To achieve better predictive performance in unsupervised learning, deep learning can be used. In deep learning, algorithms learn from different levels of representations to find the complex patterns in the data.

Any algorithm used in supervised or unsupervised learning has its own inductive bias. Inductive bias of learning algorithms refers to the set of assumptions that a learning algorithm uses for predicting the output of unseen inputs [9]. In other words, it is a set of assumptions that is used by a learning algorithm to prefer one hypothesis over the other hypothesis in the search space, in order to find a suitable hypothesis which can provide better predictions for the problem in question. These factors affect the classifier performance [10]. In the case of malware classification, an algorithm may be more accurate in classifying viruses than adware. Due to these reasons, detection results may vary from one learning algorithm to another learning algorithm. Therefore, it may be appropriate to join the results of classifiers trained at different representations to achieve the improved accuracy in predicting the class of unseen instances.

2.1 Terminology

2.1.1 Data set

A piece of information from a particular object such as a binary file or an image in a specific format is called a feature. The format of the feature is called feature representation. Each instance present in the original data is represented by its specific features for the ML processable data sets. For the malware detection task, different features can be used to represent the binary files such as files may be converted into the hexadecimal code [11, 12] or assembly instructions [13] to produce the data sets. Features may

also be extracted from the binary files such as printable strings or system calls [11] to generate the data set.

2.1.2 Feature Selection

To improve the accuracy of ML algorithms, complexity of the data sets is reduced. For this purpose, the most common approach is to apply a feature selection algorithm, which measures the quality of each feature and prioritize features accordingly [10]. Only features that can provide the valuable information for the classification are kept, and the rest are discarded. In the malware detection, an extensive feature set is produced from the binary files data set. This feature set contains many invaluable features, which can degrade the performance of a ML algorithm. Therefore, it is necessary to obtain a subset of valuable features by applying the feature selection.

2.1.3 Classification

In ML, the classification is divided into two stages, i.e., training and testing. Learning algorithms are applied on the training set to build a model (commonly called classifier) [9]. This stage is called training. During the testing stage, the generated classifier is used to predict the class of unseen instances.

2.1.4 Ensemble

Ensemble are capable of combining multiple models for the improved accuracy [10]. The different models for the ensemble may be generated from the same base algorithm on different subsets of the data or different algorithms on the same data set. Ensembles perform better than a single model due to the diversity of base models [14].

2.1.5 Trust

Trust is primarily related to the human behavior of believing a person to meet expectations. Trust has different meanings in different contexts. Trust can be defined as *"Trust is quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context"* [15]. For the problem in hand, the terms trustor and the trustee refers to an algorithm that quantify the trust. Trust can be quantified as +1 or -1; the increased or decreased value can assist in determining the extent of the trust. The trust can be quantified as the single trustor or the group trust. The trust especially the group trust is quantified for different computational problems such as for the authentication, in the peer-to-peer networks, in the mobile ad-hoc networks, for resisting the different network attacks and for spam emails.

2.2 Related Work

A significant amount of research for classification tasks has applied techniques ranging from statistical methods to machine learning like supervised learning and deep learning (DL). The use of DL has been investigated to learn different levels of the representation in order to model complex relationships in the data to classify patterns and objects [16]. DL has also been used to extract features and represent them at different layers with different representations or abstraction to be used for vision, face recognition, and hand written digit recognition. Similarly, the layered architecture has also been used for detecting the malicious behavior [2]. In some cases, the decision from an individual model or even from several models may be insufficient to obtain the final predication, especially when the cost of misclassifying one

class is higher than misclassifying the other class. As a solution, a few researchers have used veto voting for automated identification of a disease pulmonary embolism [17] and authorship identification [18].

For the malware detection, several researchers have used ensemble based on the majority voting. The majority voting is compared with different ensemble methods on five different malware data sets [19]. The majority voting is also used to generate the rules, according to the Dempster-Shafer theory, to classify the malicious codes based on the *n*-gram features of the binary files [20]. Some researchers have applied the variation of majority voting such as the weighted majority voting for the malware detection [21]. The concept of the veto voting is not investigated for the malware detection. Therefore, it is worth to investigate the veto voting for the malware detection.

To support the veto voting, the concept of trust may be used. In an early work on the authentication in open networks, the trust is used to accept or reject an entity for a task [22]. A set of inference rules is used to determine the value of trust, i.e., $0 \leq trust \leq 1$ and derived value is further used for the decision. Both direct and group trusts are used. In a study, trust value is used for resisting the non-trivial attacks on the authentication of the origin of the message in the distributed system [23]. A quantitative framework based on the attack values for resisting the attack is proposed [24, 25]. The proposed framework uses the group trust metric and calculates a trust value for all the nodes simultaneously. The values are further used to build a distributed name server, verify the meta-data in peer-to-peer networks, and resistance to the Spam e-mails. The authors also present a real world example, i.e., Advogato website[1]. A common trust based algorithm for the peer-to-peer network is the EigenTrust algorithm [26]. The EigenTrust algorithm uses peer nodes to assign the trust to each node. The assigned trust is used to computes global trust values in a distributed manner and node-symmetric manner. Global trust value is also used to distinguish malicious nodes in the system. The priority is given to the opinion of high reputation nodes. The EigenTrust algorithm is used to proposed a non-manipulable trust system for peer-to-peer networks [27]. The authors propose a partitioning technique that is used to partition the peers in groups and incentives for the peers to share files. Mobile ad hoc networks are decentralized networks and nodes in such network cooperate with each other [28]. The trust value of each node is used to improve the security of network. The authors use the local trust and recommendation trust, which are combined to obtain the combination trust. Finally, the combination trust value is used to evaluate the level of risk for ongoing tasks. In the field of multi-agent systems and ML, trust is used to make a reputation system for auction systems [29]. A generic ML based trust framework is proposed to calculate the trust of a transaction by an agent. The trust is calculated on the basis of previous successful transactions based on distinguishing features of successful and unsuccessful transactions. The distinguishing features are given as input into ML algorithms to extract the relationship. The extracted relationships are further used to predict about the success of the current transactions.

3 Veto-based Classification

In certain situations, the recommendation from more than one expert may be required. In such cases, a committee of experts is formed as it is expected that a committee always performs better than a single expert. Normally the committee uses the majority voting for combining the decisions of experts to reach a final conclusion. In some cases, the committee may grant the right to veto the decision of the committee to any member. In ML, multiple algorithms can be used to generate multiple classifiers (experts) for a classification task. Every classifier has its own inductive bias, which affects the predictive performance. Research results indicate that ensemble perform better than single classifier in fields of text categoriza-

[1] http://www.advogato.org/

tion [30] and data classification, etc. Several rules such as majority voting, i.e., bagging [10], weighted combination where weights represent effectiveness of member classifiers such as boosting [10], dynamic classifier selection [31, 32] and the veto voting [17, 18] can be used for combining the decisions and having a final prediction. Veto voting is used to give importance to a single expert (classifier) who predicts against the majority.

In malware detection, ignoring the correct prediction about a malicious file from a single expert may incur a higher cost in terms of security violations. The security violations may cause serious data loss, privacy or monetary damages to the user. Therefore, a veto voting based classification model is more appropriate than a majority voting based model. A model is proposed, which combines the inductive biases of individual classifiers and the final decision is given on the basis of veto voting. For brevity, the veto voting based classification system is referred to as veto classifier in later sections.

3.1 Voting Rules

The main objective of veto based classification is to combine the multiple predictions to reach a final decision. Formally, a veto based classification system consists of candidate classifiers (C) set and vote set V. The set of candidate classifiers C, and the set of votes (V) is a finite set (C, V) with predetermined fixed number of maximum classifiers and votes. The vote from each classifier is considered on the basis of rules given below. Some rules mentioned below are also recommended for the general voting [33].

Anonymity All votes in the vote set (V, a finite set with a predetermined number of votes) are treated equally, and the outcome of the classifier committee remains consistent with any permutation of the votes.

Neutrality All candidates in the classifier set (C, a finite set with a predetermined number of classifiers) are considered equally without any additional weighting.

Independence Candidate classifiers are independent of each other, and the outcome of the voting system remains consistent with any combination of classifiers with votes, i.e., $(C_1, V) \cup (C_2, V) \subseteq (C, V)$ where C_1 and C_2 are different combinations of classifiers.

Completeness All votes from the classifiers are considered and counted only once.

Veto Any vote indicating an instance as malware, alone can determine the outcome of the classification task regardless of the number of other votes.

Consistency The result of the voting remains consistent even if the base classifiers are split into two disjoint sets and each set vote separately. The votes from each subset create a single vote set. Formally this can be mentioned as $(C, V_1) \cap (C, V_2) \subseteq (C, V)$ where V_1 and V_2 are the partitions of votes.

Usability Voting procedure can be used for other similar problems.

Verifiability Any user of the voting system can verify the outcome of voting by counting the votes manually.

Ideally veto based voting performs better than the single classifier because if any classifier in the committee predicts the class of an instance as malware, it may veto all the other predictions from the other classifiers. For the neutrality, the results from all the classifiers are combined without any additional weighting or filtering. It is also possible to use weighting method [34] for the votes such as by assigning more weight to the vote of a classifier who outperformed all other classifiers in terms of accuracy during the training stage.

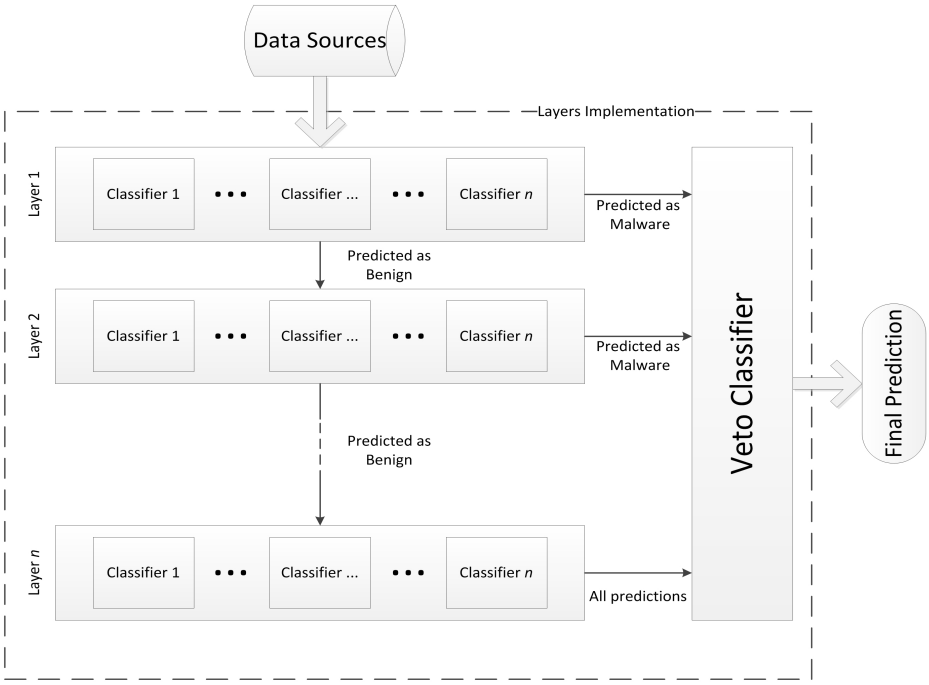


Figure 1: *N*-layer Implementation

3.2 Architecture

The model can be implemented in two possible ways, i.e., *N*-layers implementation and parallel implementation. *N*-layers implementation is based on the serial implementation.

N-Layers implementation The model can be implemented in *n*-layers with any permutation of classifiers, see Figure 1. Each layer can be customized with different *n*-gram sizes, several feature representations, various feature selection algorithms and learning algorithms. It is recommended that different classifiers shall be used while maintaining their neutrality as much as possible to increase the effectiveness [14]. From the lower layer, the instances that are declared as benign are given to the upper layer for the reclassification. In each layer, all classifiers give their predictions about the instances from the lower layer (or data sources). If at any layer, an instance is classified as malware, it is not fed into the next layer. The classification results from all the layers are given to the veto classifier. The malware prediction at any layer for any instance may be considered as veto for that particular instance. However, the final decision about the class of particular instance is taken by the veto classifier.

Parallel Implementation Instead of using layers, all the possible permutations of classifiers can be implemented in a distributed or parallel manner. Each learning algorithm is trained over the entire data set for generating the classifiers. Instead of testing only positive instances by some classifiers, each classifier works independent of results from other classifiers. All the votes from the classifiers are collected at a central location where the veto classifier outputs the final prediction.

4 Pre-processing Techniques

The selection of representative features affects the predictive performance of a classifier. Consequently, two *n*-gram extraction techniques are proposed. Most of the research studies for the malware detection demonstrate the use of either the hexadecimal-based *n*-gram data set or the opcode-based *n*-gram data set for an experiment. The proposed extraction techniques can be used to extract both representations. For this study the opcode *n*-gram extraction is performed, therefore, the proposed techniques are explained in the context of opcode *n*-grams only.

Traditionally the *n*-gram extraction is performed by using a fixed size window with a fixed step; the step size is equal to the window size. The fixed size window traverse the data file to extract the specific size *n*-grams. The generated output contains adjacent *n*-grams. To explain this process, assume that a disassembled binary file contains the following given data. A pair of characters represents an opcode (or a hexadecimal code). The task is to create *bi*-grams, i.e., *n*-gram of size 2 from this data file.

aa bb cc dd ee ff gg hh ii jj kk ll mm nn oo pp

The generated *bi*-grams from this file are "*aabb ccdd eeff gghh iijj kkll mmnn oopp*" and so on. The fixed size window is unable to extract some *n*-grams such as "*bbcc*" or "*ddee*". If the file size is large and the data is redundant then there is a probability to have missing combinations, but still missing *n*-grams cannot be produced in the appropriate frequency and can have less importance for the classification task.

4.1 Overlapping *n*-grams

To address the problems of missing *n*-grams, the use of configurable sliding window to generate overlapping *n*-grams is proposed. The window can be configured with two parameters, i.e., size and step. The size parameter defines the size of a *n*-gram to be extracted, and the step parameter defines the number of opcodes to be skipped before extracting the next *n*-gram. It is expected that all possible combinations can be extracted by this extraction technique. Referring the example in Section 4, if the window is configured as following, size = 2, i.e., two adjacent opcodes are extracted to form a *n*-gram and step = 1, i.e., after the *n*-gram extraction window skips one opcode (first opcode) to move forward. This configuration generates "aabb *bbcc* ccdd *ddee* eeff *ffgg* gghh *hhii* iijj *jjkk*" and so on.

4.2 Non-adjacent Opcode Extraction

Either by using the traditional *n*-gram extraction or the overlapping *n*-gram extraction, extracted *n*-grams can provide the information only about the dependencies of the adjacent opcodes. It is valuable to look at the information provided by non-adjacent opcodes. Non-adjacent opcodes have dependencies such as they can be function header and tail. Some changes are proposed in the overlapping *n*-gram extraction method to explore the information both from non-adjacent opcode and non-adjacent adjacent opcode. The size parameter is changed to the start-end size parameter. The start-end size parameter defines the number of adjacent opcodes to be extracted for the start and the end of a *n*-gram. The step size parameter defines the number of opcodes to be skipped for extracting a new *n*-gram. A new parameter is introduced, i.e., gap size, which specifies the interval between start and end opcode or number of opcodes to be skipped between the start and the end opcode of a *n*-gram. The example mentioned in the Section 4 can be used to describe this procedure. If the window is configured as following for extracting non-adjacent *bi*-grams, start-end size = 1, i.e., one opcode for the start and one opcode for the end of a *n*-gram are extracted, step = 1 and gap = 1, i.e., one opcode between the start opcode and the end opcode of a *n*-gram is skipped. This configuration produces the *bi*-grams, which contains non-adjacent opcodes.

Table 1: Experiment with one data set and three algorithms

Data Set[a]	Algorithm	TP	TN	FP	FN	R[b]	P[b]	F1[b]
***n*-gram**	JRip	243	184	66	07	0.972	0.786	0.869
	J48	226	225	25	24	0.904	0.900	0.902
	IB*k*	224	225	25	24	0.896	0.899	0.897
	Veto[c]	243	203	47	07	0.972	0.837	0.900
	Majority[c]	223	233	17	26	0.895	0.929	0.912
Overlap	JRip	238	197	53	12	0.952	0.817	0.879
	J48	232	234	16	18	0.928	0.935	0.931
	IB*k*	224	224	26	26	0.896	0.896	0.896
	Veto	246	208	42	04	0.984	0.854	0.914
	Majority	230	240	10	20	0.920	0.958	0.938
S. Window	JRip	209	215	35	41	0.836	0.856	0.846
	J48	215	205	45	35	0.860	0.826	0.843
	IB*k*	164	237	13	86	0.656	0.926	0.768
	Veto	242	139	111	08	0.968	0.685	0.802
	Majority	220	204	46	30	0.880	0.827	0.852

[a] The full names of data sets are *n*-gram data set, overlap data set and sliding window data set.

[b] R is Recall, P is Precision, and F1 is F-Measure.

[c] Veto is Veto Classifier and Majority is Majority voting.

[d] Veto Classifier and Majority voting both are applied on all the three data sets.

The generated output is "*aacc bbdd ccee ddff eegg*" and so on. To have non-adjacent adjacent opcodes in a *n*-gram, the configuration can be changed as follow: start-end = 2, i.e., two adjacent opcodes for the start and the end of a *n*-gram are extracted; the step size and the gap size are kept 1. The generated output is "*aabbddee bbcceeff ccddgghh*" and so on. If the value of the gap size and the step size parameters is changed from 1 to 2, the generated output is "*aabbeeff ccddgghh eeffiijj*" and so on.

5 Experiment I

The aim of the experiment is to evaluate the proposed veto voting based malware detection method and impact of the proposed data pre-processing techniques and compare the results with the majority voting. The proposed method can be used to detect either a specific type of malware or different types of malware; however in this study a single family of malware is used. The experimental data set contains Windows-based executable files. Windows is a common OS for novice users and contains different vulnerabilities[2], which can be exploited by a malware. When a binary file in the data set is disassembled, different file features such as assembly language instructions and printable strings, are produced in the text format, which are further processed to extract the assembly directives, i.e., opcode. Opcodes are further processed to produce the *bi*-gram data sets using different strategies. Different text categorization techniques can be applied to the output generated in the previous step to get discriminating features of benign and malware. Term Frequency-Inverse Document Frequency (*tf-idf*) is used to derive the significant features from the data sets. The extracted features are used to create Attribute-Relation File Format (ARFF)[3] files. ARFF file is a structured ASCII text file that includes a set of data instances, each described by a set of features [10]. ARFF files are used as input to the proposed model, which

[2] http://technet.microsoft.com/en-us/security/bulletin/

[3] http://www.cs.waikato.ac.nz/ml/weka/arff.html

uses Waikato Environment for Knowledge Analysis (Weka) application programming interface (API) [35] for applying learning algorithms to build and analyze classifiers. A pre-experiment is performed for the selection of learning algorithms. The first experiment is divided into two sub-experiments. In the first experiment, the inductive biases of the different classifiers built on the same data set are combined. Second experiment combines the inductive biases of individual classifiers built on different data sets. In both experiments, the results from all the classifiers are given to the veto classifier for the final prediction.

5.1 Feature Representation

Opcode is used for generating *bi*-grams as features. It is concluded in the previous studies that opcode *n*-grams are better choice for the malware detection in comparison to other features such as printable strings, systems calls or byte code (hexadecimal) *n*-grams [13]. The opcode *n*-grams are capable of providing the information about the program flow, structure and function that cannot be deduced from other representations.

5.2 Data Set Creation

For the experiment, scareware (rouge) software is selected as malware representation. The reason for this choice is, there is a subtle difference between scareware and benign. In case of traditional malware, presence of malicious payload distinguishes a malware from the benign. However, in scareware no specific malicious payload is available that can be used to differentiate a scareware from the benign. Absence of malicious payload may deceive human expert for the classification of a particular software as scareware.

Scareware are scam software that usually masquerade as an anti-virus software and resembles the benign software in functionality. Scareware generates the false alarm about the presence of malware in the user's machine. The false alarms are used to scare the users into disclosing their credit card information for buying the protection[4]. No public data set e.g., virus, Trojan, and worm data sets is available for the scareware detection experiments. Therefore, a data set with 500 files is created; out of which 250 files are scareware, and 250 files are representing benign. The benign files are default applications of Windows OS such as notepad, paint and applications available online for download at CNET Download[5]. All the benign files are scanned with commercial security software (anti-malware) to reduce the chances of malware presence in a benign file. Scareware files are obtained from the malware database of Lavasoft[6].

5.3 Pre-Processing

The disassembled file is a standard text file, which contains three fields, i.e., the memory address of the instruction, the byte-based location in the file and the instruction itself (combination of opcode and operands). The next step is to extract only opcodes from the files and discard irrelevant information, i.e., operands. The extracted opcodes are saved in the original order. After opcodes extraction from the disassembled files, three different procedures are used to tokenize the data to produce *bi*-grams for three different data sets. Each row in a data set represents a *bi*-gram, i.e., concatenation of two opcodes. Hereafter, these three data sets are referred to as *bi*-gram data set, overlap data set and sliding window data set respectively to indicate the method used in creating that particular data set. The *bi*-gram size has yielded the best performance in a previous study [36] and possible combinations of opcodes to produce *bi*-grams are limited, depending upon the number of reserve words in the assembly language.

[4]http://news.bbc.co.uk/2/hi/8313678.stm
[5]http://download.com
[6]http://lavasoft.com

For the *bi*-gram data set, a fixed size window traverse each input file from top to bottom. In every step, a *n*-gram consisting of two opcodes is extracted and recorded in another file having the similar file name, but different extension. The purpose of keeping the similar name is to keep track of benign files and scareware files, so each file can be represented at the same position in all three data sets and finally in the ARFF file. For overlap data set method mentioned in the Section 4.1 is followed with the configuration, i.e., size = 1 and step = 1. For the sliding window data set, start-end size and step parameters are kept one. To obtain the nonadjacent opcode *bi*-grams, each file is processed in four consecutive passes with a gap size ranging from 1-4. Due to the changing gap size, the first generated *bi*-grams are having a gap of one opcode between the start opcode and the end opcode, in the second pass there is a gap of two opcodes and so on. The process of generating the sliding window data set is slower than generating the *bi*-gram data set and the overlap data set. However, the computational cost and memory requirements for generating the sliding window data set are lower than creating large size *n*-grams.

5.4 Feature Selection

Many real world problems are complex. To apply learning algorithms, the dimensionality of the complex problem is reduced by choosing a subset of significant features from the given set of (raw) features. The selected subset of features plays significant role in the increase/decrease of either classification and/or computational performance. Significant feature selection is done by using a feature selection algorithm, removing features that are deemed unlikely to improve the classification process. In the field of text classification, *tf-idf* shows promising results for the valuable features selection. In this experiment *tf-idf* is applied on data sets to limit the number of features to top 1000 features per data set. The *tf-idf* is a statistical measure of importance of a *bi*-gram in the entire data set [37]. The *tf* is the number of times a *bi*-gram occurs in a file; *df* is the number of files in a class that contain a specific *bi*-gram. The *idf* of a *bi*-gram is obtained by dividing the total number of files (N) by the *df* and then taking the logarithm.

5.5 Performance Evaluation Criteria

Each learning algorithm is evaluated by performing cross-validation tests. Confusion matrices are generated by using the responses from the classifiers. The following four estimates define the elements of a confusion matrix: True Positive (TP) represents the correctly identified scareware programs. False Positive (FP) represents the incorrectly classified benign programs. True Negative (TN) represents the correctly identified benign programs, and False Negative (FN) represents the incorrectly identified scareware programs. The performance of each classifier is evaluated using Recall (R), which is the ratio of scareware programs correctly predicted from the total number of scareware programs, Precision (P), ratio of scareware programs correctly identified from the total number of programs identified as scareware. F-Measure (F1) is the harmonic mean of the precision and the recall and is the final evaluation measure.

5.6 Pre-Experiment for Algorithm Selection

A number of studies have addressed the similar problem with different learning algorithms; however, none of the authors is conclusive on the choice of algorithms either for the malware detection or according to the produced data set. In a number of studies Ripper (JRip) [38], C4.5 Decision Tree (J48) [39], *k*-nearest neighbor (IB*k*) [40] Naive Bayes [10] and SMO [41] outperformed other algorithms. Based on previous research, a pre-experiment is performed to evaluate all these algorithms on all the three data sets. The top three algorithms, i.e., JRip, J48 and IB*k* are considered as candidates, to combine their inductive biases for the final prediction in the proposed model.

5.7 Results and Discussion

In the first experiment, one data set is used to build classifiers from three different algorithms. In the second experiment, three data representations are used and one classifier is trained from each representation. Majority voting is compared to the veto voting. In the first experiment, the predictions from three classifiers are collected and given to the veto classifier and the majority voting. The predictions from all the classifiers including both voting strategies on each data set are shown in Table 1. In the second experiment, three algorithms, i.e., JRip for *n*-gram data set, JRip for the overlap data set, and J48 for the sliding window data set are selected on the basis of the recall in the first experiment. These algorithms are used to built three classifiers and the predictions about each instance from these classifiers is given to the veto classifier and the majority voting for the final prediction. The results of this experiment (see Table 2) indicate that the recall of the veto classifier is better than the recall values in the first experiment. Majority voting shows the similar behavior for the precision.

The experimental results indicate that combining the inductive biases of different algorithms trained on multiple representations predicts better for the malware detection than combining the inductive biases of different algorithms trained on the same data set. The experimental results of both voting strategies can be discussed in three dimensions by using three measures, i.e., recall, precision, and f-measure. The experimental results show that the veto classifier has better recall than the majority voting, i.e., veto classifier reduces the number of misclassified scareware. Recall is the key measure as the objective of the veto approach is to reduce the likelihood of malware misclassification while tolerating a percentage of false positives or decrease in the precision. If the system is tuned to predict all applications as malware, it will produce a high false positive rate, which is undesirable from a user's point of view. Users need the accurate prediction both for the malware and benign applications. Therefore, the precision is also considered as a complimentary measure with the recall. The veto classifier shows a higher tendency for the correct detection of scareware while the majority voting shows a tendency towards the detection of benign applications. Therefore, the precision rate is higher for the majority voting. There are few instances, which are misclassified by both voting schemes. Most of these instances are benign, but predicted as scareware by both the veto classifier and the majority voting. However, the number of such instances is minimal. The precision and the recall have an inverse relationship if the precision increases, the recall decreases. Therefore, another evaluation measure is required, which combines the precision and the recall. Thus, the final evaluation measure is F-measure, which evenly weights the precision and the recall. It may be argued that the arithmetic mean of the precision and the recall can also be used as a composite measure. However, the arithmetic mean is an inappropriate measure as with 100 % R and 0 % P or vice versa; the arithmetic mean is always 50 %. This is not the case with the harmonic mean as the harmonic mean is always less than or equal to the arithmetic mean [42]. If there is a difference between the value of R and the P such that the value of R is significantly smaller than P, the harmonic mean tends strongly towards the recall. F1 of the majority voting is higher than the veto classifier, which favors the use of majority voting for the problem in question. However, the recall for the majority voting, which is a key measure, is lower than the veto classifier so it may be argued that the veto is a better approach for the malware detection. Thus, the veto classifier shall be extended to increase the precision.

Bi-grams are used as the feature in the experiment because they are computationally inexpensive to produce. Generally such short combinations may not represent an important function or set of instructions in the files and are difficult to analyze. However, *bi*-grams in the sliding window data set can provide the valuable information for the scareware analysis due to the combination of non-adjacent opcodes. Scareware resembles the benign applications such as displaying popup windows or alert messages, and showing the dialog boxes. Therefore, it is difficult for the human experts to predict about the scareware by analyzing the functionality of an application only. The proposed model helps the human

Table 2: Experiment with three data sets and one algorithm on each data set

Data Set[a]	**Algorithm**	**TP**	**TN**	**FP**	**FN**	**R**[b]	**P**[b]	**F1**[b]
n-gram	JRip	243	184	66	07	0.972	0.786	0.869
Overlap	JRip	238	197	53	12	0.952	0.817	0.879
S. Window	J48	215	205	45	35	0.860	0.826	0.843
	Veto[d]	248	195	55	02	0.992	0.818	0.896
	Majority[d]	223	247	03	26	0.895	0.986	0.938
	Trust Veto[e]	235	230	20	15	0.940	0.922	0.931

[a] The full names of data sets are *n*-gram data set, overlap data set and sliding window data set.

[b] R is Recall, P is Precision, and F1 is F-Measure.

[c] Veto is Veto Classifier and Majority is Majority voting.

[d] Veto Classifier,Majority voting and Trust-based veto Classifier are applied on all the three data sets.

[e] Trust Veto is Trust-based Veto Classifier.

expert by automating the process of analyzing and predicting the scareware (malware). JRip and J48 algorithm are considered expensive algorithms in terms of time consumed to train and generate the model. However, it is easy to analyze the rules and trees generated to differentiate the scareware and benign.

The decisions of the different classifiers are combined to produce better results, and such combination shall not be considered as a substitute of a good classifier [43]. The proposed veto classifier follows the same principle. Veto classifier is neither a substitute of a good classifier nor replacing the majority voting. In the domain of decision theory, it has been suggested that different voting strategies shall be adopted for different tasks according to the problem in question. We argue that the veto classifier is a better choice for the malware detection task as this approach addresses the problems of the majority voting. There are different problems related with the majority voting such as majority voting may ignore the right decision of the minority. While ignoring the decision from the minority votes, the total number of majority votes may have an ignorable difference in comparison with the total number of minority votes. Another problem of the majority voting is the choice of the number of candidate classifiers. If the number of selected classifiers is an odd, then a simple majority can be obtained, but if the number of selected classifiers is an even then a situation may arise where equal numbers of votes are given to both the benign and malware classes. In the domain of ML, different variations of majority voting has been suggested such as restricted majority voting, enhanced majority voting, and ranked majority voting to address the problems of majority voting[44]; such problems are avoided with the proposed veto classifier.

The results of the veto classifier depend upon a suitable permutation of the algorithms. Some permutations may obtain 100 % recall by just predicting all applications as malware. Some permutation can achieve 100 % precision, if all the instances are predicted as benign applications. Before permutation, classifiers selection is a critical and complex task. For a small number of classifiers, an optimal combination can be found exhaustively, but as the number of classifiers increases, the complexity of selection is increased due to their different inductive biases, search limits and practical applicability. The classifier selection process can be improved by a static selection or dynamic selection method [45].

6 Experiment II

Results of the experiment I in the Section 5.7 suggest that the veto classifier is a better choice for the malware detection problem. Results also indicate that the majority voting skews towards benign ap-

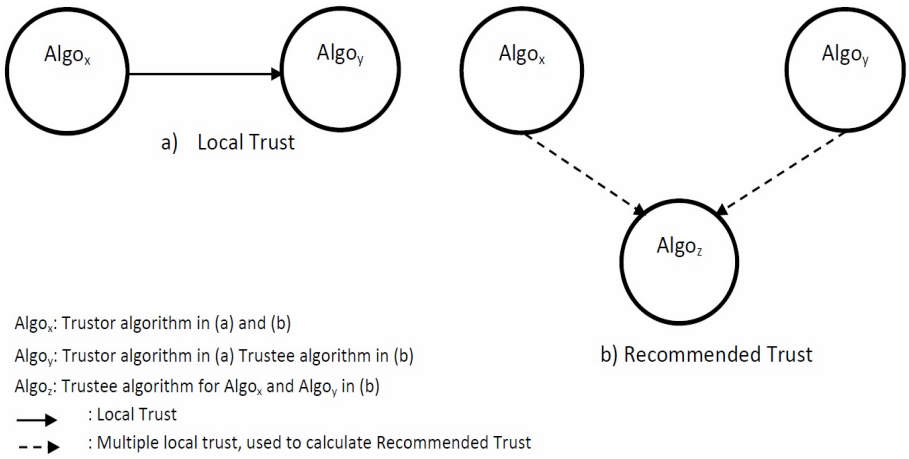


Figure 2: The direct trust and the recommended trust of algorithms

plications and the veto classifier skews towards malicious applications. In the Experiment I, all algorithms are treated equal for the final prediction to assure the "neutrality" property for the veto classifier. Consequently, algorithms who generally demonstrate higher misclassification in comparison to other algorithms (generally referred to as weak learners), supersede the correct prediction. This phenomenon produces a high false positive rate and the precision of the veto classifier is significantly lower than the majority voting. To address the above mentioned problems in the veto classifiers, use of the algorithm's "trust" is suggested and the veto classifier is extended as a trust-based veto classifier. The aim of this experiment is to evaluate the trust-based veto classifier for the malware detection and compare the performance of the proposed algorithm with the majority voting and the veto classifier.

6.1 Trust-based Veto algorithm

Trust as a quantitative measure can be quantified with integer values. The positive integer may represent the trust while a negative integer may be used to represent the distrust. The quantified trust value in computational problems can be calculated for participating nodes, algorithms, and agents. For the machine learning problems, different kinds of trust can be calculated for algorithms contending in the system. Consequently, an algorithm is proposed, which generate trust-based veto classifier for the malware detection. The trust-based veto algorithm involves three kinds of trust, i.e., local trust (can also be referred to as direct trust), recommended trust, and global trust. Each algorithm in the system calculates its trust level for other algorithms in the system, i.e., how much $algorithm_x$ trusts the $algorithm_y$ in terms of predicting the class of an instance, called local trust (t), see Figure 2. The local trust value is further used to calculate the recommended trust (RT) for each algorithm, see Figure 2. The recommended trust aids in calculating the global trust value (GT) of each algorithm. The global trust value is used for having a veto decision.

6.1.1 Local Trust Calculation

Local trust of an $algorithm_x$ on $algorithm_y$ $(t_y : algo_x \rightarrow algo_y)$ is calculated by comparing the predictions (d) of both algorithms with each other and the actual class (C) of the instance, see Algorithm 1. Suppose, from a data set of benign and malicious instances, an instance of benign class is given to the $algorithm_x$ and the $algorithm_y$ for predicting the class of the instance. There is a finite set of possible

predictions, i.e., both algorithms may predict correct, or both algorithms may predict incorrect, or any one of the algorithms may predict the correct class. If both algorithms have the same prediction for the instance, either correct or incorrect; trust is not affected. However, if the $algorithm_x$ predicts the incorrect class and $algorithm_y$ predicts the correct class, the $algorithm_x$ increases the trust level (*sat*) of the $algorithm_y$ with +1. In case, the $algorithm_x$ predicts the correct class and the $algorithm_y$ predicts incorrect class, the $algorithm_x$ increase the distrust level (*unsat*) of the $algorithm_y$ with +1. All the instances in the data set are given to both algorithms sequentially for the prediction. At the end of process, local trust of the $algorithm_y$ is calculated by dividing trust (*sat*) with the sum of the trust (*sat*) and the distrust (*unsat*), see Algorithm 1.

Algorithm 1 Trust Calculation

Require: Actual Class of Instance (C), prediction of $algo_x$ (d_x), prediction of $algo_y$ (d_y)

```
function LOCALTRUST
    repeat
        if dx = dy then                    ▷ Prediction of both algorithms may be correct or incorrect
            movenext
        end if
        if dx ≠ dy then                    ▷ Compare the prediction with the C
            if dx = C then
                unsat ← unsat(algo_x, algo_y) + 1
            else
                dy ≠ C
                sat ← sat(algo_x, algo_y) + 1
            end if
        end if
    until !EOF
end function
(t_y : algo_x → algo_y) ← sat(algo_x,algo_y) / (sat(algo_x,algo_y)+unsat(algo_x,algo_y))
```

6.1.2 Recommended Trust Calculation

The local trust shows the unique trust on a particular algorithm (e.g., $algo_y$) from another algorithm (e.g., $algo_x$). This value varies from the algorithm to the algorithm in the system and cannot be used as a final metric for deciding about a veto in the system. The Recommended trust is calculated to address this problem. The local trusts on an algorithm from all the other algorithms in the system are summed to calculate the recommended trust. The recommended trust value represents the combine trust of all algorithms in the system on that particular algorithm. If the set of all algorithms is $S = \{algo_0, algo_1, algo_2, \ldots, algo_n\}$ then we may have a two subsets $S' = \{algo_0\}$ and $S" = \{algo_1, algo_2, \ldots, algo_n\}$. The subset $S"$ is having all the algorithms in the system as members except the algorithm $algo_0$ for which the RT is calculated. The algorithm $algo_0$ is the member of the subset S'. The RT is calculated by using the Equation (1).

$$RT_y \leftarrow \sum_{n=1}^{n} (t_y : algo_n \rightarrow algo_y) \qquad \forall algo_n \in S" \tag{1}$$

6.1.3 Global Trust Calculation

The *RT* varies from algorithm to algorithm and may not be compared on the similar scale. Consequently, *RT* of an algorithm is normalized to obtain the global trust of that particular algorithm. The term normalization represents distinct, but related meanings in different contexts. The basic purpose of normalization is to convert the different values on a notionally standard scale to compare them equally with each other. The normalized *GT* value lies in the interval of the [0-1] and is calculated by using the Equation (2):

$$GT_y \leftarrow \frac{RT_y}{\sqrt{\sum_{n=1}^{n} RT_n^2}} \tag{2}$$

6.1.4 Veto Decision

The calculated *GT* value is used for deciding a veto for the prediction of a set of algorithms by another algorithm or set of other algorithms. Suppose a system in which seven algorithms are participating for predicting the class (benign or malicious) of an instance that belongs to the malicious class. A subset *M* of four algorithms in the system predicted the class of the instance as benign, and a subset *V* of three algorithms predicted the class of the instance as malicious. The mean of both groups is calculated. If the mean of *V* is greater than the mean of *M*, the *V* can veto the decision of the *M* and the outcome will be the prediction of the *V*.

However, for this experiment, there is a change in the veto decision function due to less algorithms. The change is explained as following. There are three algorithms in the system, i.e., $algo_x$, $algo_y$, and $algo_z$. If two algorithms, i.e., $algo_x$, and $algo_z$ classify the instance as a benign and only one algorithm, i.e., the $algo_y$ classify the instance as a malware; the $algo_y$ can veto according to Equation (3). The change in the prediction strategy is to reduce the random decision errors.

$$Veto : GT_y \geq \frac{GT_x + GT_z}{2} \tag{3}$$

6.2 Results and Discussion

The classifiers combination to form an ensemble can be divided roughly into two categories, i.e., multiclassifier and multirepresentation [4]. In the multiclassifier approach, a set of classifiers is trained on the same representation of the data. In the multirepresentation approach, different classifiers are trained on the multiple representations. On the basis of the experimental results presented in the Section 5.7, the trust-based veto classifier is applied only for combining the inductive biases of several algorithms trained on the different representations. The experimental results are shown in the Table 2. Experimental results indicate two issues. First, the low TN of the veto classifier that leads to a high FP; the low TN is because of veto classifier's skew towards malware. Second, the low TP of the majority voting that leads to a high FN; the low TP is because of the majority voting's skew towards benign programs. The trust-based veto classifier performs better than the veto classifier in terms of TN and reduces the FP. The trust-based veto classifier is also better than the majority voting in terms of TP and reduces the FN. However, TP of trust-based veto classifier is better than the majority voting and less than the veto classifier. Recall of the trust-based veto classifier is better than the majority voting and less than the veto voting. In terms of F-Measure, the difference between the values of f-measure of the majority voting and the trust-based veto classifier is minimal, so one can argue that the trust-based veto classifier is an optimal choice for the malware detection due to inherited skewness towards the malware. The majority voting and the veto classifier are computationally inexpensive as the prediction from each algorithm is counted for the outcome. In trust-based veto classifier, each algorithm evaluates the trust and maintains

the trust info locally in a trust table without significantly increasing the processing overhead; however, the storage requirement is higher than the majority voting and the veto classifier as they do not store any information. The locally stored trust information is provided to the system for the decision purpose, when required. Trust-based veto classifier provides a direct experience of the trust on each algorithm. Due to direct experience, there is no central authority for maintaining the trust information, which makes the proposed algorithm a self-policing algorithm.

One property mentioned in the voting rule set is anonymity (see section 3.1). The trust-based veto classifier maintains the anonymity property as all votes are treated equally and no vote is discarded. However, the veto property is not followed as it is mentioned in section 3.1 for the trust-based veto classifier experiment. In the changed veto decision strategy, a single algorithm indicating the instance as the malware cannot affect the outcome of the detection task. Now for the veto, algorithm or set of algorithms need to meet certain criteria, which reduced the chances of errors and terminate the prediction of weak learners. However, with the proposed strategy, there is a probability that the prediction of weak learner/s may be always ignored, if the trust on that particular algorithm is significantly less than the trust on other algorithms. Suppose a detection system with five algorithms where two algorithms are weak learner with significantly low trust levels. This particular group of algorithms may not veto the decision of all other algorithms for all the cases, even if the prediction was correct. However, the changes in the veto strategy will increase the robustness of trust-based classifier as any number of algorithms can compete for the veto decision with any number of algorithms.

To follow the veto strategy mentioned in the section 3.1, one alternative direction is to allocate the trust to each algorithm on the basis of predetermined criteria such as previous performance. The trust of all algorithms may be readjusted regularly on the basis of prediction performance. All the algorithms vote for the decision. For the veto decision, when an algorithm predicts the instance as malware the trust of that particular algorithm can be compared with a specific threshold to obtain the final decision. However, it is worth to note that veto strategy does not perform as expected for the encrypted malware, i.e., malware with the encrypted malicious routine. The encrypted part of the malware cannot be disassembled to obtain accurate instruction sequences or byte code. The presence of encryption in a file can be considered as the indication of the malicious behavior. The encrypted malware can be decrypted or executed to decrypt in a controlled environment to obtain the data files. The data files can be further disassembled to extract instruction sequences or byte code.

7 Conclusion and Future Work

There are a several strategies to obtain the result of an ensemble such as the majority voting and the veto voting. However, it is not investigated which decision strategy is optimal for the malware detection. Most of the researchers have used the majority voting for the malware detection. A veto-based classification was proposed that was able to predict about malware better than the majority voting. A series of experiments with *n*-gram data sets, generated from different strategies, were performed. A recent threat, i.e., scareware was used as malware representation. The results indicated that the proposed model reduced the number of false negatives (malware detected as legitimate application), however, the false positive of proposed model was very high. The decision strategy of proposed model was improved, i.e., trust-based veto classifier. The experimental results indicated that the improved classifier perform better than the previous approach in terms of the false positive rate. The proposed trust-based veto classifier performed better in the recall than the majority voting. However, for the composite measure F1, the majority voting was slightly better than the trusted-veto classifier and the trusted veto classifier was better than the veto

voting. The experimental results also indicated the suitability of each voting scheme for detecting a particular class of software. For the future work, the aim is to further improve the proposed model in two different directions, i.e., improvement in the selection of classifiers for the optimal results, and parameter tuning of the selected classifiers. The proposed model will also be tested for the detection of different types of malware and for the multi-class prediction.

References

[1] R. K. Shahzad and N. Lavesson, "Veto-based malware detection," in *Proc. of the 7th International Conference on Availability, Reliability, and Security (ARES'12), Prague, Czech Republic*. IEEE, August 2012, pp. 47–54.

[2] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, "A layered architecture for detecting malicious behaviors," in *Proc. of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID'08), Cambridge, Massachusetts, USA, LNCS*, vol. 5230. Springer-Verlag, September 2008, pp. 78–97.

[3] Y. Sun, M. S. Kamel, and A. K. C. Wong, "Empirical study on weighted voting multiple classifiers," in *Proc. of the 3rd International Conference on Advances in Pattern Recognition and Data Mining (ICAPR'05), Bath, UK*. Springer-Verlag, August 2005, pp. 335–344.

[4] M. S. Kamel and N. M. Wanas, "Data dependence in combining classifiers," in *Proc. of the 4th International Conference on Multiple Classifier Systems (MCS'03), Guildford, UK, LNCS*, vol. 2709. Springer-Verlag, June 2003, pp. 1–14.

[5] L. Lam and S. Y. Suen, "Application of majority voting to pattern recognition: an analysis of its behavior and performance," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 27, no. 5, pp. 553–568, 1997.

[6] Y.-A. Sun and C. R. Dance, "When majority voting fails: Comparing quality assurance methods for noisy human computation environment," *Computing Research Repository*, vol. 1204.3516, 2012.

[7] C. Ren, J.-F. Yan, and Z.-H. Li, "Improved ensemble learning in fault diagnosis system," in *Proc. of the 2009 International Conference on Machine Learning and Cybernetics (ICMLC'09), Baoding, China*. IEEE, July 2009, pp. 54–60.

[8] W. Stallings, *Network Security Essentials: Applications and Standards*, 4th ed. Prentice Hall, 2011.

[9] T. M. Mitchell, *Machine Learning*, 1st ed. McGraw-Hill, Inc., 1997.

[10] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann Inc., 2011.

[11] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. of the 2001 IEEE Symposium on Security and Privacy (S&P'01), Oakland, California, USA*. IEEE, May 2001, pp. 38–49.

[12] R. K. Shahzad, S. I. Haider, and N. Lavesson, "Detection of spyware by mining executable files," in *Proc. of the 5th International Conference on Availability, Reliability, and Security (ARES'05), Krakow, Poland*. IEEE, February 2010.

[13] R. K. Shahzad and N. Lavesson, "Detecting scareware by mining variable length instruction sequences," in *Proc. of the 10th Annual Information Security South Africa Conference (ISSA'11), Johannesburg, South Africa*. IEEE, August 2011, pp. 1–8.

[14] L. I. Kuncheva, "Diversity in multiple classifier systems," *Information Fusion*, vol. 6, no. 1, pp. 3–4, 2005.

[15] T. W. A. Grandison, "Trust management for internet applications," Ph.D. dissertation, Imperial College, 2003.

[16] A. Gepperth, "Object detection and feature base learning with sparse convolutional neural networks," in *Proc. of the 2nd International conference on Artificial Neural Networks in Pattern Recognition (ANNPR'06), Ulm, Germany, LNCS*, vol. 4087, August 2006, pp. 221–232.

[17] D. Tikk, Z. T. Kardkovács, and F. P. Szidarovszky, "Voting with a parameterized veto strategy: Solving the KDD cup 2006 problem by means of a classifier committee," *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations*, vol. 8, no. 2, pp. 53–62, 2006.

[18] R. Kern, C. Seifert, M. Zechner, and M. Granitzer, "Vote/veto meta-classifier for authorship identification - notebook for pan," in *Proc. of the Conference on Multilingual and Multimodal Information Access Evaluation (CLEF'11), Amsterdam, Netherlands*, September 2011.

[19] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," *Computational Statistics & Data Analysis*, vol. 53, no. 4, pp. 1483–1494, 2009.

[20] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang, "Malicious codes detection based on ensemble learning," in *Proc. of the 4th International Conference on Autonomic and Trusted Computing (ATC'07), Hong Kong, China, LNCS*, vol. 4610. Springer-Verlag, July 2007, pp. 468–477.

[21] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici, "Unknown malcode detection via text categorization and the imbalance problem," in *Proc. of the 2008 International Conference on Intelligence and Security Informatics (ISI'08), Taipei, Taiwan*. IEEE, June 2008, pp. 156–61.

[22] T. Beth, M. Borcherding, and B. Klein, "Valuation of trust in open networks," in *Proc. of the 3rd European Symposium on Research in Computer Security (ESORICS'94), Brighton, UK, LNCS*, vol. 875. Springer-Verlag, November 1994, pp. 3–18.

[23] M. K. Reiter and S. G. Stubblebine, "Path independence for authentication in large-scale systems," in *Proc. of the ACM Conference on Computer and Communications Security (CCS'97), Zurich, Switzerland*. ACM, April 1997, pp. 57–66.

[24] R. Levien, "Attack resistant trust metrics," UC Berkeley, Tech. Rep., 2004.

[25] ——, "Attack-resistant trust metrics," in *Computing with Social Trust*. Springer-Verlag, 2009, pp. 121–132.

[26] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *Proc. of the 12th International Conference on World Wide Web (WWW'03), Budapest, Hungary*. ACM, May 2003.

[27] Z. Abrams, R. McGrew, and S. Plotkin, "A non-manipulable trust system based on eigentrust," *Special Interest Group on Electronic Commerce Exchanges*, vol. 5, no. 4, pp. 21–30, 2005.

[28] X. Li, J. Slay, and S. Yu, "Evaluating trust in mobile ad hoc networks," in *Proc. of the Workshop of International Conference on Computational Intelligence and Security (CIS'05), Xi'an, China, LNCS*, vol. 380. Springer-Verlag, December 2005.

[29] X. Liu, G. Trédan, and A. Datta, "A generic trust framework for large-scale open systems using machine learning," *Computing Research Repository*, vol. 1103.0086, 2011.

[30] S. M. Weiss, C. Apte, F. J. Damerau, D. E. Johnson, F. J. Oles, T. Goetz, and T. Hampp, "Maximizing text-mining performance," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 4, pp. 63–69, 1999.

[31] G. Giacinto and F. Roli, "Methods for dynamic classifier selection," in *Proc. of the 10th International Conference on Image Analysis and Processing (ICIAP'99), Venice, Italy*. IEEE, September 1999, pp. 659–664.

[32] A. Santana, R. G. F. Soares, A. M. P. Canuto, and M. C. P. d. Souto, "A dynamic classifier selection method to build ensembles using accuracy and diversity," in *Proc. of the 9th Brazilian Symposium on Neural Networks (SBRN'06), Ribeirão Petro, Brazil*. IEEE, October 2006, pp. 36–41.

[33] F. Talib, "Computational aspects of voting: A literature survey," Master's thesis, Rochester Institute of Technology, Rochester, USA, 2007.

[34] H. Moulin, "Voting with proportional veto power," *Econometrica*, vol. 50, no. 1, pp. 145–62, 1982.

[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations*, vol. 11, pp. 10–18, 2009.

[36] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using OPCODE representation," in *Proc. of the 1st European Conference on Intelligence and Security Informatics (EuroISI'08), Esbjerg, Denmark, LNCS*, vol. 5376. Springer-Verlag, December 2008, pp. 204–215.

[37] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Managment*, vol. 24, no. 5, pp. 513–523, August 1988.

[38] W. W. Cohen, "Fast effective rule induction," in *Proc. of 12th International Conference on Machine Learning (ICML'95), California, USA*. Morgan Kaufmann Publishers Inc., July 1995, pp. 115–23.

[39] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[40] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[41] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Advances in Kernel Methods-Support Vector Learning*, vol. 208, pp. 98–112, 1999.

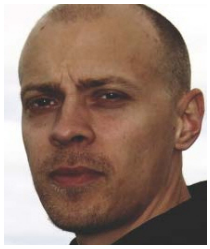
[42] R. H. Walpole, Ronald E.and Myers and S. L. Myers, *Probability & Statistics for Engineers & Scientists*. Prentice Hall, 2012.

[43] J. Franke, L. Lam, R. Legault, C. P. Nadal, and C. Y. Suen, "Experiments with the CENPARMI data base combining different classification approaches," in *Proc. of the 3rd International Workshop on Frontiers in Handwriting Recognition (ICFHR'93), Buffalo, New York, USA*, May 1993, pp. 305–311.

[44] A. F. R. Rahman, H. Alam, and M. C. Fairhurst, "Multiple classifier combination for character recognition: Revisiting the majority voting system and its variations," in *Proc. of the 5th International Workshop on Document Analysis Systems (DAS'02). Princeton, USA, LNCS*, vol. 2423. Springer-Verlag, August 2002, pp. 167–178.

[45] D. Ruta and G. Bogdan, "Classifier selection for majority voting," *Information Fusion*, vol. 6, no. 1, pp. 63–81, 2005.

Raja Khurram Shahzad is PhD student at Blekinge Institute of Technology in Karlskrona, Sweden. He received his M.Sc. in Security Engineering, in 2009 from Blekinge Institute of Technology. His main area of research is Machine Learning with a special focus on application of supervised learning algorithms. Current research interests involve the application of supervised learning to potentially unwanted programs detection.



Niklas Lavesson is Associate Professor of Computer Science at Blekinge Institute of Technology in Karlskrona, Sweden. He received his M.Sc. in Software Engineering and Ph.D. in Computer Science, in 2003 and December 2008 respectively, from Blekinge Institute of Technology. His main area of research is Machine Learning with a special focus on analysis and evaluation of supervised learning algorithms. Current research interests involve the application of supervised learning to data mining problems in healthcare management and document analysis. Lavesson is the Coordinator of Research Education at Blekinge Institute of Technology.