J Comput Virol (2006) 2:231–239
DOI 10.1007/s11416-006-0027-8

ORIGINAL PAPER

N-gram analysis for computer virus detection

D Krishna Sandeep Reddy · Arun K Pujari

Received: 10 August 2006 / Revised: 19 September 2006 / Accepted: 1 October 2006 / Published online: 8 November 2006

Abstract Generic computer virus detection is the need of the hour as most commercial antivirus software fail to detect unknown and new viruses. Motivated by the success of datamining/machine learning techniques in intrusion detection systems, recent research in detecting malicious executables is directed towards devising efficient non-signature-based techniques that can profile the program characteristics from a set of training examples. Byte sequences and byte *n*-grams are considered to be basis of feature extraction. But as the number of *n*-grams is going to be very large, several methods of feature selections were proposed in literature. A recent report on use of information gain based feature selection has yielded the best-known result in classifying malicious executables from benign ones. We observe that information gain models the presence of *n*-gram in one class and its absence in the other. Through a simple example we show that this may lead to erroneous results. In this paper, we describe a new feature selection measure, *class-wise document frequency* of byte *n*-grams. We empirically demonstrate that the proposed method is a better method for feature selection. For detection, we combine several classifiers using Dempster Shafer Theory for better classification accuracy instead of using any single classifier. Our experimental results show that such a scheme detects virus program far more efficiently than the earlier known methods.

D. Krishna Sandeep Reddy · A. K. Pujari (✉)
Artificial Intelligence Lab,
University of Hyderabad, Hyderabad 500 046, India
e-mail: dksr@inbox.com

A. K. Pujari
e-mail: akpcs@uohyd.ernet.in

1 Introduction

A *computer virus* is a code that recursively replicates a possibly evolved copy of itself. Viruses infect a host file or system area, or they simply modify a reference to such objects to take control and then multiply again to form new generations [22]. Any computer system is vulnerable to malicious code whether or not it is attached to other systems. Recent *Gartner* report ranked viruses and worms as top security threats [8] and detecting malicious code has become one of the prime research interests in the field of information security. A majority of commercial antivirus products rely on *Signature based virus detection* and a heuristic classifier that detects new virus. The 'classic' signature based detection algorithm uses signatures of known viruses to generate detection models. Signatures create a unique tag for each virus so that future examples of it can be correctly classified with a small error rate. These methods do not generalise well to detect unknown and unseen viruses. Unknown or new viruses will easily escape the detection by simple defenses like code obfuscation, as their signatures are not present in the database [5]. Moreover, as the number of known viruses increases, the size of the signature database increases and also the time of checking a file for virus signatures increases [13].

In order to overcome these disadvantages, data mining approaches [25] are proposed recently for detection of computer virus. The success of data mining techniques have been demonstrated in the context of intrusion detection systems. The main idea is to have a supervised classification scheme where the known instances of virus and benign programs are considered as a training set and a set of feature extracted from the training set are used to build a classification model to

profile these two classes—benign and virus instances. Data mining approaches are based on the assumption that viruses have certain typical characteristics in common and these characteristics are not present in benign programs. For instance, virus writers may use some generic virus generating toolkits to write and compile their code and hence the viruses generated using this toolkit have certain common features that are specific to the toolkit, the compiler and also the programming environment. Thus any data mining based virus scanner is expected to be more robust. Of course, the performance of such a method critically depends on the set of features and on the classifier. The generic approach is to formulate the detection problem as a supervised classification and to decide upon a set of features that are useful for classification. Normally *n*-grams of byte sequence of the programs are considered as the basic features and several classifiers such as IBk, TFIDF, Naive Bayes, SVM, decision trees are studied for classification. Most often no single classifier can yield satisfactory classification and hence there are also attempts to combine multiple classifiers in the present context. Recent combination frameworks based on the learning set's sampling (*bagging, arcing, boosting*) or the features selection subsets showed an interesting way to reduce correlation errors amongst weakened individual classifiers and then reduce the misclassification rate. We give a brief review of the research on data mining approach for detection of virus programs in the next section.

In this paper we propose a very elegant method of detection of computer viruses by extracting the common features of known virus programs. We introduce a concept of *relevant n-grams* and present a new feature selection technique based on class-wise document frequency and relevant *n*-grams. We make use of multiple classifiers and use Dempster Shafer Theory of Evidence for combining classifiers for the classification task. We demonstrate that our feature selection measure performs better than information gain, which was used in the earlier work [11]. The motivation of the the present work is the observation that the earlier proposals of feature selection of *n*-grams do not adequately model the profile of both the classes. We discuss this aspect detail in the following section. Our training set consists of two classes, virus executables class and benign executables class, where each class have 250 instances. The viruses were simple Portable Executable (PE) infectors with little obfuscation and do not change their code much unlike poly/metamorphic viruses. Most of the benign executables were taken from system32 folder in Windows. For viruses, we used only the loader programs; we did not use any infected programs in our analysis.

The rest of the paper is organised as follows. In Sect. 2 we review earlier research on data mining approach to detect malicious executables. We observe that most of the approaches choose byte *n*-grams as the features and as there can be very large number of *n*-grams, several feature selection methods are also adopted. We illustrate these techniques using a motivational example. This example also brings out the situations where the earlier feature selection methods do not adequately model the benign and virus classes of programs. In Sect. 3, we introduce the concept of relevant *n*-grams and present our method of feature selection. Section 4 outlines the process of combining multiple classifiers. The experimental results are discussed in Sect. 5. We show that the proposed method performs better than the best known method.

2 Related Work

Theoretically, the problem of detection of malicious executables in general (and virus detection, in particular) is known to be a hard problem [4]. Several techniques have been proposed in literature for detection of malicious codes. These attempts can be classified into two categories—*signature-based* and *non-signature-based*. Non-signature based techniques [28] are normally based on data mining and machine learning principles where the detection system models behaviour of the process or the user based on earlier-known data. The research on malicious code detection can also be classified based on the type of malicious code that are to be detected. Kephart et al. [10] propose the use Neural Networks to detect boot sector malicious binaries. Using an ANN classifier with all bytes from the boot sector malicious executables as input, it is shown that unknown boot sector malicious executables can be successfully identified with low false positive rate. Later, Arnold et al. [2] apply the same techniques to Win32 binaries. Motivated by the success of data mining techniques in host based and network intrusion system, Schultz et al. [19] propose several data mining techniques to detect different types of malicious executables. In a companion paper [18] the authors develop a UNIX mail filter that detects malicious windows executables based on the above work. Byte sequences are used as the set of features as machine codes are most informative to represent executables. The RIPPER algorithm of rule discovery and Naive Bayes classifiers are used for classification in this study. Abou-Assaleh et al. [1] observe that

n-grams extracted from byte sequences can be used as an effective feature. They use Common *n*-gram (CNG) as the features and propose a *k*-NN classification for detecting computer virus. Kolter et al. [11] independently realise that *n*-grams can possibly be used as a set of features. However, as the set of all *n*-grams is very large, it is proposed to use few of them selected based on their information gain. Kolter et al. [11] also investigate several classification techniques, which are implemented in WEKA [25] and boosted J48 algorithm reportedly gave good results. We restrict our study here only to non-signature based techniques for detection of virus programs [10,11,19,28].

It is also interesting to note that many of these techniques use byte sequences and some of these studies propose byte *n*-grams as the basic features to detect malicious codes. Byte *n*-grams are overlapping substrings, collected in a sliding-window fashion where the windows of fixed size slides one byte at a time. The concept of byte *n*-gram is inspired by commonly known concept of word *n*-gram. Word *n*-grams and character *n*-grams are widely used in natural language processing, information retrieval and text mining [3,9]. It is observed that *n*-grams not only capture the statistics of substrings of length *n* but implicitly represent frequencies of longer substrings as well. *N*-grams have the ability to capture implicit features of the input that are difficult to detect explicitly. Byte *n*-grams can be viewed as features in the present context when an executable program is viewed as a sequence of bytes. Importance of byte *n*-gram in detecting computer virus has been realised more than once in the computer virology research. In 1994, a byte *n*-gram-based method is used for automatic extraction of virus signatures [10]. The major difficulty in considering byte *n*-grams as a feature is that the set of all byte *n*-grams obtained from the set of byte strings of virus as well as of the benign programs is very large and it is not useful to apply classification techniques directly on these. So restricting to a smaller set of relevant *n*-grams is necessary for classification the viruses and benign executables. There are several feature selection techniques proposed in [27] and the important ones are *document frequency* and *information gain*. In the present context the *document frequency* is the number of executables in which a *n*-gram occurs and *information gain* measures the number of bits of information obtained for category prediction by knowing the presence or absence of a *n*-gram in a program. In [11], features are chosen to be set of all sequences of four bytes and a feature selection method based on *Information Gain* is used. *N*-grams were also used for malware phylogeny generation [7], where the authors attempt to build better models.

Definition *Let C be a class of executable programs. C is a subset of the training set. The information gain of the n-gram Ng is given by*

$$\mathrm{IG}(Ng) = \sum_{v_{Ng}\in\{0,1\}} \sum_{C\in\{C_i\}} P(v_{Ng}, C) \log \frac{P(v_{Ng}, C)}{P(v_{Ng})P(C)} \quad (1)$$

where C is one of two classes-benign or virus, v_{Ng} is the value of n-gram Ng; $v_{Ng} = 1$ if n-gram Ng occurs in the program and $= 0$, otherwise. $P(v_{Ng}, C)$ is the proportion of programs in C in which that n-gram Ng takes on value v_{Ng}. $P(v_{Ng})$ is the proportion of programs in entire training set such that n-gram Ng takes the value v_{Ng}. $P(C)$ is the proportion of the training data belonging to the class C.

The *n*-grams are sorted in decreasing order of information gain and top L *n*-grams are considered for building training data, where L is the profile length.

In a different but related context [1], a feature selection method based on *average term frequency* is used and the *Common N-Gram analysis* (CNG) method is used. The CNG classification method relies on profiles for class representation. During training, the data for each class is collected and *n*-grams with their normalized term frequencies are counted.

Definition *For a given n-gram Ng and a program P, the term frequency $\tau(Ng, P)$ of Ng in P is the frequency of occurrences of Ng in P. The normalised term frequency is defined as $\frac{\tau(Ng,P)}{\tau}$, where τ is $\sum_{P\in C} \sum_{Ng\in P} \tau(Ng, P)$.*

The normalised frequency of each *n*-gram is computed with respect to both the classes of programs and are sorted in decreasing order. Top $L/2$ *n*-grams are selected from each class to constitute a profile of L *n*-grams. In order to understand these two approaches and to illustrate their ability to detect virus programs, we consider following motivating example.

Example

Let us assume that the training set consists of 20 executables of which there are 10 virus programs and 10 benign programs. Let there be six *n*-grams Ng_i, $i = 1, \ldots, 6$, extracted from these programs. Tables 1 and 2 give the term frequencies of the *n*-grams in virus and benign programs respectively. Table 3 shows the class-wise document frequencies of these *n*-grams in virus and benign programs. In the Table 3, for example, Ng_1 is present in 2 out of 10 viruse programs and in 9 of 10 benign executables (i.e. value '1' means that the *n*-gram is present and the value '0' means that the *n*-gram is not present).

The information gain for the *n*-gram Ng_1 is calculated as follows using Eq. 1.

Table 1 *N*-grams and their term frequency for virus class

Virus	Ng_1	Ng_2	Ng_3	Ng_4	Ng_5	Ng_6
p1	4	0	0	0	2	1
p2	3	3	0	0	1	2
p3	0	6	0	0	1	2
p4	0	1	0	0	2	1
p5	0	0	0	0	2	1
p6	0	0	0	5	0	0
p7	0	0	5	4	0	0
p8	0	0	6	0	0	0
p9	0	0	0	0	0	0
p10	0	0	0	0	0	0

Table 2 *N*-grams and their term frequency for benign class

Benign	Ng_1	Ng_2	Ng_3	Ng_4	Ng_5	Ng_6
p11	2	1	0	0	0	4
p12	1	1	0	2	0	3
p13	3	1	0	2	3	2
p14	1	2	1	1	7	0
p15	2	1	3	1	4	0
p16	4	3	2	0	0	0
p17	1	2	3	3	0	0
p18	1	0	1	0	0	0
p19	1	1	1	0	0	0
p20	0	0	2	0	0	0

Table 3 *N*-grams and their class-wise document frequencies

Class	*Value*	Ng_1	Ng_2	Ng_3	Ng_4	Ng_5	Ng_6
V	1	2	3	1	2	5	6
V	0	8	7	9	8	5	4
B	1	9	8	7	6	3	3
B	0	1	2	3	4	7	7

$$\begin{aligned}\mathrm{IG}(Ng_1) &= P(v_{Ng_1}=1, C=V)\log\frac{P(v_{Ng_1}=1, C=V)}{P(v_{Ng_1}=1)P(C=V)} \\ &+ P(v_{Ng_1}=0, C=V)\log\frac{P(v_{Ng_1}=0, C=V)}{P(v_{Ng_1}=0)P(C=V)} \\ &+ P(v_{Ng_1}=1, C=B)\log\frac{P(v_{Ng_1}=1, C=B)}{P(v_{Ng_1}=1)P(C=B)} \\ &+ P(v_{Ng_1}=0, C=B)\log\frac{P(v_{Ng_1}=0, C=B)}{P(v_{Ng_1}=0)P(C=B)}\end{aligned}$$

Substituting the probability values that we can be calculated from Table 3, we get

$$\begin{aligned}\mathrm{IG}(Ng_1) &= \frac{2}{10}\log\frac{\frac{2}{10}}{\frac{11}{20}\cdot\frac{1}{2}} + \frac{8}{10}\log\frac{\frac{8}{10}}{\frac{9}{20}\cdot\frac{1}{2}} + \frac{9}{10}\log\frac{\frac{9}{10}}{\frac{11}{20}\cdot\frac{1}{2}} \\ &+ \frac{1}{10}\log\frac{\frac{1}{10}}{\frac{9}{20}\cdot\frac{1}{2}} = 2.6568\end{aligned}$$

In the similar manner the information gain for all 6 *n*-grams can be calculated as $\mathrm{IG}(Ng_2) = 2.3823, \mathrm{IG}(Ng_3) = 2.5916, \mathrm{IG}(Ng_4) = 2.2490, \mathrm{IG}(Ng_5) = 2.0606$, and $\mathrm{IG}(Ng_6) = 2.1332$.

The top 4 *n*-grams with high information gain are Ng_1, Ng_3, Ng_2 and Ng_4. We see (Table 1) that Ng_5 and Ng_6 appear more frequently in large proportion of virus programs than any other *n*-grams. Thus these two *n*-grams are more appropriate to represent the virus programs but are ignored by the information gain based feature selection.

In the CNG method, the best *n*-grams are selected based on their class wise average term frequency. Table 1 lists the term frequencies of the *n*-grams in the virus class and we see that highest normalised term frequencies correspond to Ng_3 and Ng_2. Similarly, from Table 2, the top 2 *n*-grams for benign class are Ng_1 and Ng_5. If we compute the common *n*-grams for the above example and select four most frequent (in terms of normalised term frequency) *n*-grams, two from each class, then we get Ng_3 and Ng_2 from virus class and Ng_1 and Ng_5 from benign class.

3 Relevant *n*-grams

In this section we discuss our proposal for feature selection. We assume that we have a set of executables that are known to be viruses and another set of executables that are known to be benign. Our aim is to identify a set of features that are common to the set of viruses and similarly another set of features that are common to the benign executables. Let D be the training set containing a set of virus programs V and a set of benign programs B, i.e., $D = B \cup V$. Our feature extraction method is based on *n*-grams and the feature selection measure is a variant of document frequency. We introduce here two novel concepts—*relevant n-grams* for a class and *class-wise document frequency*.

Definition *For n-gram Ng, the document frequency* $\delta(Ng, P)$ *of Ng with respect to a program P is 1 if Ng is present in P and 0, otherwise. The classwise document frequency of Ng with respect to a class C is* $\delta(Ng, C) = \sum_{P\in C}\delta(Ng, P)$. *In other words, the classwise document frequency is the number of executable programs in C that contain Ng.*

While the document frequency is a kind of global measure the classwise document frequency is a local measure with respect to a class. The main advantage of classwise document frequency is that we can analyze each class independently. Moreover, the number of distinct *n*-grams in a program is far smaller than that for the

entire training set and hence it saves memory requirements as we deal with only *n*-grams of one class at a time. We propose a novel concept of *relevant n-grams* for a class D of executable programs. For each executable program, we find the set of all *n*-grams and let Ng^t be the set of all *n*-grams for a program $t \in D$.

Definition *The set of all n-grams for D,* $Ng(D)$ *is* $\cup_{t\in D} Ng^t$. *We assume that the elements of* $Ng(T)$ *are arranged in the non-increasing order of class-wise document frequency for each class V and B. Thus we have two non-increasing lists,* $\delta(Ng_i, V)$ *and* $\delta(Ng_i, B)$. *Let* V_r (B_r) *be the set of top* $k(= L/2)$ *n-grams selected from the set* $\delta(Ng_i, V)$ ($\delta(Ng_i, B)$, *respectively). We define* $Ng_k(D)$ *as the set of relevant n-grams with respect to D and is* $V_r \cup B_r$.

We give in Fig. 1 the flowchart for the relevant *n*-grams selection.

Example (contd.)

Table 3 gives the classwise document frequency of the *n*-grams. We see that top 2 *n*-grams of each class based on class-wise document frequency are Ng_1, Ng_2, Ng_5 and Ng_6. Based on the justification given above, this is a better combination of *n*-grams to adequately represent both the classes.

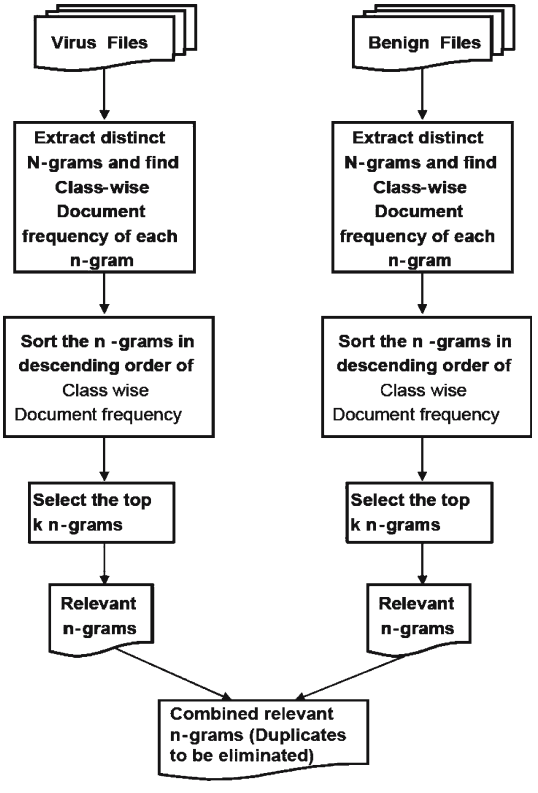

Fig. 1 Flow chart for selecting relevant *n*-grams set from virus and benign executables' hexdump data

Once the set of relevant *n*-grams are selected, we use the vector-space model of Information Retrieval to represent the set of programs D in terms of $Ng_k(D)$. A program is represented as a vector of terms $t_1, t_2, \ldots, t_m$ where $t_i (1 \leq i \leq m)$ is a binary (0–1) value denoting the single or multiple occurrences of ith *n*-gram $Ng_k(D)$ in the program. Thus when a program is viewed as a vector each unique relevant *n*-gram corresponds to a dimension. The value 1 represents the occurrences of the *n*-gram in the program and its absence is represented by 0. Our training set consists of a set of labelled vectors—the vector representation of the set of programs together with the respective class label [virus(V) or benign(B)].

4 Classification methodology

As we observed in the previous section, the detection problem reduces essentially to a supervised classification problem. Several algorithms exist [14] for supervised classification like Support Vector Machine, decision tree, neural networks etc. Different classifiers reveal different possibilities for separating the data. In order to harness the complementary information and merits of individual classifiers, we can combine the classifiers to improve the performance of classification. The main motivation for combining classifiers is that the sets of patterns misclassified by the different classifiers would not necessarily overlap. There are several approaches for combining classifiers [6]. Three broad groups of combining classifiers are *parallel combining* of classifiers computed for different feature sets, *stacked combining* of different classifiers computed for the same feature space, and *combining weak classifiers* trained on modified versions of the original dataset [6]. In the present work we use Dempster Shafer Theory of Evidence for combining classifiers [29].

4.1 Dempster–Shafer theory of evidence (D–S theory)

The D–S Theory is an alternative to traditional probabilistic theory for the mathematical representation of uncertainty [17]. An important tenet of this theory is the combination of evidence obtained from multiple sources and the management of conflict between them. D–S Theory starts by assuming a *Universe of Discourse* θ, also called a *Frame of Discernment*, which is a set of mutually exclusive alternatives.

A function $m : 2^\theta \rightarrow [0, 1]$ is called a basic probability assignment if it satisfies $m(\phi) = 0$ and $\sum_{A\subseteq\theta} m(A) = 1$.

The summary of $m(B)$ for all subsets $B \subseteq A$ becomes the total belief in A and Bel(A) is a measure of the total belief committed to A.

$$\mathrm{Bel}(A) = \sum_{B \subseteq A} m(B) \tag{2}$$

Two independent evidences expressed as two basic probability assignments m_1 and m_2 can be combined into a single basic assignment $m_{1,2}$ by D–S rule of combination as given below.

$$m_{1,\,2}(A) = \begin{cases} \dfrac{\sum\limits_{B \bigcap C = A} m_1(B) m_2(C)}{1 - \sum\limits_{B \bigcap C = \phi} m_1(B) m_2(C)}, & A \neq \phi \\ 0, & A = \phi \end{cases}$$

4.2 Combining classifiers using D–S theory

The D–S rule of combination works better in combining evidences but does not give intuitive results when the conflict among evidences is more [15]. In such a case, we can discount the sources first and then combine the resulting functions with D–S rule of combination (or an alternative rule) using a discounting function [17,21]. Shafer applies the discounting function to each specified *Belief*.

Let $1 - \alpha$ be the degree of confidence of a particular belief function, A, where $0 \leq \alpha_i \leq 1$ and i is an index used to specify the particular discounting factor for a particular belief measure. $\mathrm{Bel}_{\alpha_i}(A)$ then represents the discounted belief function [20] given by:

$$\mathrm{Bel}_{\alpha_i}(A) = (1 - \alpha_i)\mathrm{Bel}(A) \tag{3}$$

We then average all the belief functions associated with set A $(\mathrm{Bel}_{\alpha_i,1}(A), \mathrm{Bel}_{\alpha_i,2}(A), \mathrm{Bel}_{\alpha_i,n}(A))$ to obtain aggregated belief measure $\overline{\mathrm{Bel}}$.

$$\overline{\mathrm{Bel}}(A) = \frac{1}{n}(\mathrm{Bel}_{\alpha,1}(A) + \cdots + \mathrm{Bel}_{\alpha,n}(A)) \tag{4}$$

for all subsets A of the universal set X.

In the present context, each classifier k of K classifiers for a given query instance x, gives vector $\mathrm{BEL}_k(x) = (\mathrm{Bel}_{k1}(x), \ldots, \mathrm{Bel}_{ki}(x), \ldots, \mathrm{Bel}_{kI}(x))$, where $\mathrm{Bel}_{ki}(x)$ is the probability that x belongs to class i according to classifier k. We follow the 'discount and combine' model and calculate the total belief for each class using the formula $\mathrm{Bel}_i(x) = \sum_k^K \alpha_{ki}\mathrm{Bel}_{ki}(x)$. The instance x belongs to class i for which $\mathrm{Bel}_i(x)$ value is maximum.

There are three methods for finding the discount factors, α_{ki} [12], and one method among them is learning the discount factors based on the use of training set and the minimization of an error criterion. Let the data set be $\mathcal{L} = \{(y_n, x_n), n = 1, \ldots, N\}$, where y_n is the class value and x_n is a vector representing the attribute values of the nth instance. Let $\mathcal{L}_1, \ldots, \mathcal{L}_j$ be almost equal parts of $\mathcal{L}$. Define $\mathcal{L}_j$ and $\mathcal{L}^{(-j)} = \mathcal{L} - \mathcal{L}_j$ to be the test and training sets for the jth fold of a J-fold cross-validation. Given K learning algorithms, which we call *level-0 generalizers* [23], let $M_k^{(-j)}$, for $k = 1, \ldots, K$ be the model obtained by invoking the kth algorithm on the data in the training set $\mathcal{L}^{(-j)}$. These are called *level-0 models*. If model $\mathcal{M}_k^{(-j)}$ is used to classify an instance x in $\mathcal{L}_|$ and $P_{ki}(x)$ denote the probability of the ith output class, and the vector

$$\mathcal{P}_{kn} = (P_{k1}(x_n), \ldots, P_{ki}(x_n), \ldots, P_{kI}(x_n)) \tag{5}$$

gives the model's class probabilities for the nth instance, assuming that there are I classes. Assembling together the class probability vectors from all K models, along with the actual class:

$$\mathcal{L}_{CV} = (y_n, \mathcal{P}_{1n}, \ldots, \mathcal{P}_{kn}, \ldots, \mathcal{P}_{Kn}), \quad n = 1, \ldots, N. \tag{6}$$

These are the *level-1 data*. We use linear regression learning algorithm that we call the *level-1 generalizer* to derive from these data a model $\widehat{\mathcal{M}}$ for y as a function of $(\mathcal{P}_{1n}, \ldots, \mathcal{P}_{kn}, \ldots, \mathcal{P}_{Kn})$.

If the original classification problem has I classes, it is converted into I separate regression problems, where the problem for class ℓ has instances with response equal to one when they have class ℓ and zero otherwise. The linear regression for class ℓ is simply

$$\mathrm{LR}_\ell(x) = \sum_k^K \alpha_{k\ell} P_{k\ell}(x). \tag{7}$$

We choose the coefficients α_{kl} to minimize

$$\sum_j \sum_{(y_n,\, x_n) \in \mathcal{L}_j} \left(y_n - \sum_k \alpha_{k\ell} P_{k\ell}^{(-j)}(x_n) \right)^2.$$

This method of combining classifiers and finding discount factors can be readily implemented in WEKA using the StackingC metaclassifier with linear regression option. Stacking is a way of combining multiple models that have been learned for a classification task [26]. It uses a higher level model to combine lower level models for greater accuracy. The higher level model can be a Naive Bayes, Multi-response linear regression (MLR) etc. We combine Support Vector Machine (SVM), Instance-based Learner (IBK), and decision tree in WEKA using StackingC with linear regression option. To implement decision tree, we used J48 algorithm available in WEKA. The reason for choosing these classifiers is that they are entirely different from each other in their approach of classification.

5 Experimental results

There is no benchmark data set available for the detection of malicious executables unlike intrusion detection. Data sets (i.e. viruses) collected from the website VX Heavens [24] were used by [1,11]. We collected 250 viruses from [24] and 250 benign executables from our lab. All the executables were in Windows PE format. Most of the benign executables were taken from system32 folder in Windows. Each executable in the dataset is converted to hexadecimal codes in ASCII format. We extracted *n*-grams by sliding a window of different lengths one *byte* at a time. We experimented for different value of length *n* as 2, 3 and 4. The classwise document frequency for each class is determined for each *n*-gram during the process of extraction of *n*-grams. The *n*-grams are sorted in non-increasing order of their class-wise document frequency and top *k n*-grams are selected, where *k* is called profile length of each class. These *k* *n*-grams selected from each class are combined to form relevant *n*-grams set of cardinality **K** with duplicates removed. We experimented with **K** between 100 and 500. This final set of *n*-grams are the relevant *n*-grams that are useful in classifying the viruses and benign executables.

All the classification algorithms are implemented in WEKA [25]. We used the default values given by WEKA for all classifiers except for IBK, where *k* value is changed to 5 and used stratified tenfold cross-validation.

We compare our work with that of [11], where the authors obtained good results in the context of using *n*-grams for detecting malicious executables. In [11], the experiments are done on two data sets, one with 476 malicious and 561 benign executables and the other set contains 1971 malicious and 1651 benign executables. Information gain is used as feature selection measure. After extracting all *n*-grams along with the information gain value, the *n*-grams are sorted in decreasing order of information gain and top *k* *n*-grams are taken and vector space model is formed. Out of several classifiers, it is claimed that the best results are obtained for boosted J48 algorithm. In order to compare our work, we implemented their technique of using information gain as feature selection measure and J48 algorithm as classifier on our data.

In Data Mining, Receiver Operating Characteristics (ROC) curves are used to compare classification capability of different algorithms. The more the area under the ROC curve of an algorithm, the more robust and better it is in classification.

Figure 2a and 2b give the results that compare class-wise document frequency with information gain, used in [11], with the same classifier, boosted J48. From the

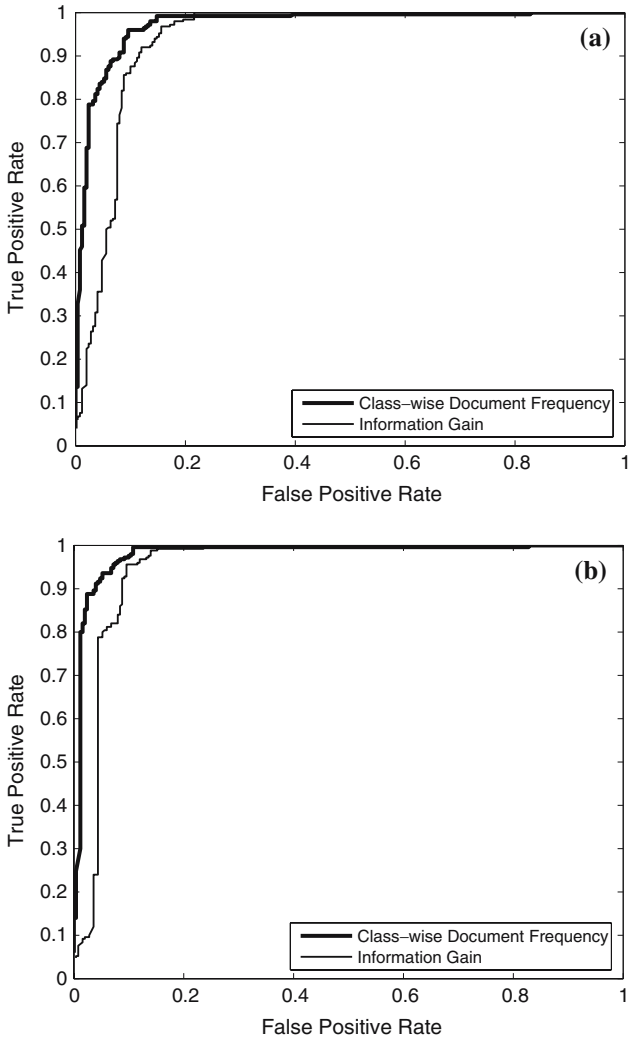

Fig. 2 ROC graphs to compare class-wise document frequency and information gain: **a** *n*-gram length is 2 and profile length is 100; **b** *n*-gram length is 4 and profile length is 100

visual inspection of these two graphs and taking area under the ROC curves into consideration, it is clear that class-wise document frequency approach outperforms that of information gain. One reason might be that in case of information gain, most of the relevant *n*-grams with highest information gain are from the benign programs. This makes the data mining algorithms to model only the benign programs and there are no representative *n*-grams for virus programs. But with class-wise document frequency we do not face this problem as we are combining the relevant *n*-grams from both the classes i.e. viruses and benign programs. This ensures that the data mining algorithms have adequate information of both the classes to correctly model both the classes. Also the advantage of class-wise document frequency is that we can analyze each class independently. This saves memory requirements as we will be dealing with only *n*-grams of one class at a time.

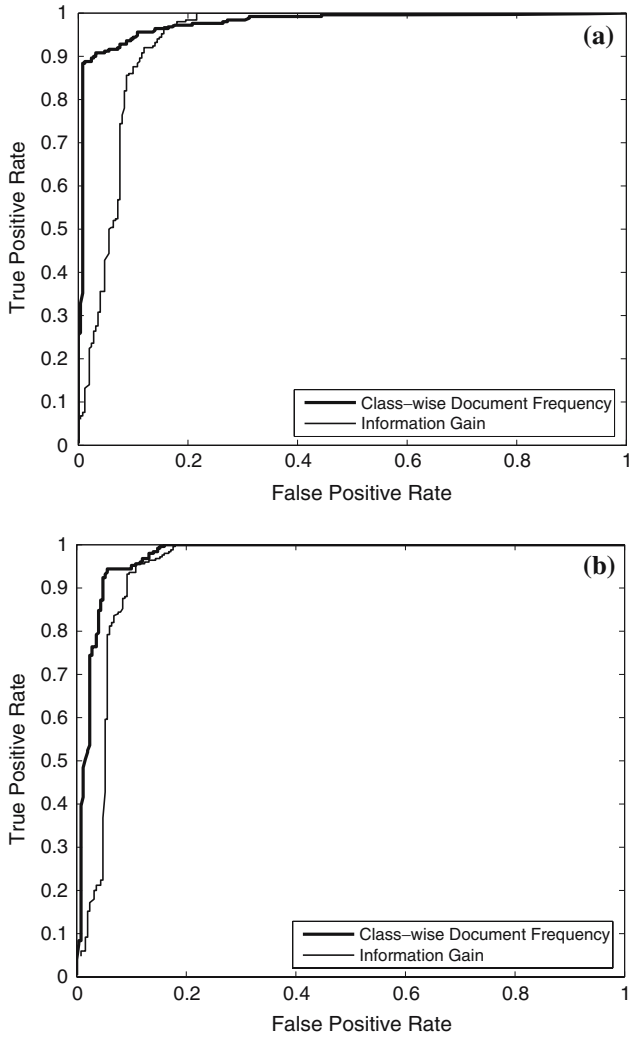

Fig. 3 ROC graphs to compare our approach with that of [11] **a** *n*-gram length is 2 and profile length is 100; **b** *n*-gram length is 3 and profile length is 100

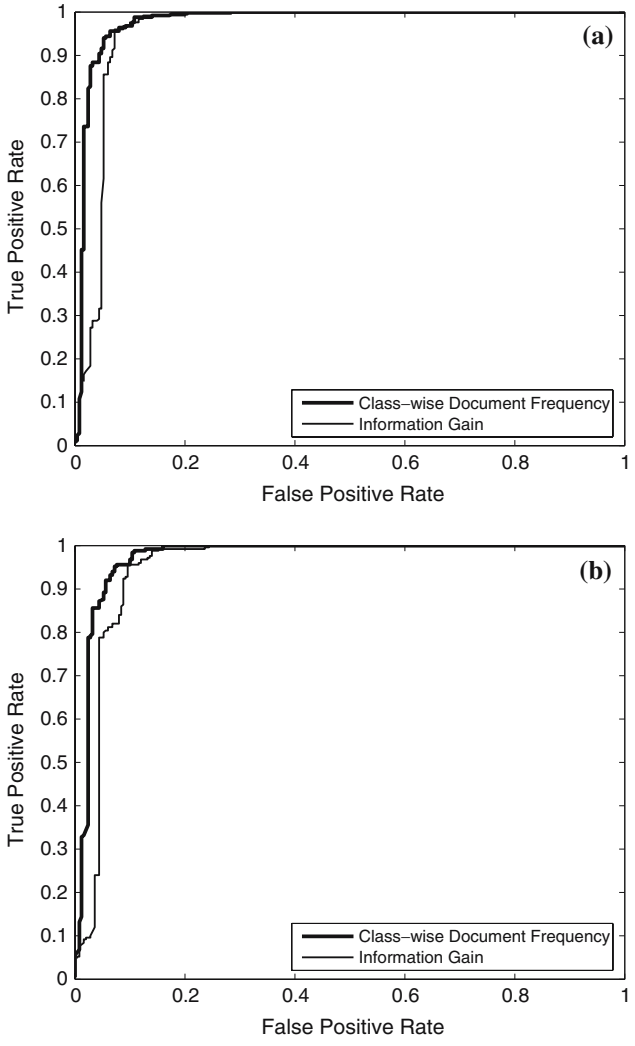

Fig. 4 ROC graphs to compare our approach with that of [11] **a** *n*-gram length is 4 and profile length is 100; **b** *n*-gram length is 4 and profile length is 500

Figures 3a and 4b give the overall result of our method compared to information gain as feature selection measure and boosted J48 classification algorithm as in [11]. The graphs are for different values of *n*-gram length and profile length of a class. In terms of area under ROC curves, it is clear from the graphs that our approach is better in all cases.

When it comes to advanced computer viruses like polymorphic viruses, static analysis methods do not work. To tackle these viruses, static analysis methods should be combined with dynamic analysis methods for efficient detection. For example, polymorphic virus consists of three components—decryption routine, mutation engine and virus body. Since the mutation engine and the virus body are encrypted and decryption routine is different in each replication, it is not possible to directly apply any static analysis method (including our method) to detect this virus. Instead we can use a dynamic analysis technique (like sandboxing) that trick a polymorphic virus into decrypting and revealing itself [16]. On this decrypted virus we can use the static analysis method. Here we assume that a polymorphic virus must decrypt before it can execute normally.

6 Conclusion

In this paper, we proposed a new feature selection measure, class-wise document frequency, and introduced relevant *n*-grams in the context of using *n*-grams for computer virus detection. For classification, we used Dempster–Shafer Theory of Evidence to combine classifiers—SVM, decision tree and IBK. Our approach outperformed the earlier work done in this context.

The *n*-gram approach lacks semantic awareness. Because of this it is very difficult to analyse the relevant *n*-grams that we obtain. Our future work will be to develop a semantic aware method and also include different kinds of malicious and benign executables in our training data. Also we are exploring the use of variable-length *n*-grams in this context.

References

1. Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.: Detection of new malicious code using n-grams signatures. In: PST, pp. 193–196 (2004)
2. Arnold, W., Tesauro, G.: Automatically generated win32 heuristic virus detection. In: Proceedings of the 2000 International Virus Bulletin Conference (2000)
3. Cavnar, W.B., Trenkle, J.M.: N-gram-based text categorization. In: Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, pp. 161–175. Las Vegas, US (1994)
4. Cohen, F.: Computer viruses: theory and experiments. Comput. Secur. **6**(1), 22–35 (1987)
5. Christodorescu, M., Jha, S.: Static analysis of executables to detect malicious patterns. In: Proceedings of the 12th USENIX Security Symposium (Security'03), pp. 169–186. USENIX Association, USENIX Association (2003)
6. Duin, R.P.W., Tax, D.M.J.: Experiments with classifier combining rules. In: MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems, London, pp. 16–29. Springer, Berlin Heidelberg New York (2000)
7. Karim, Md.E., Walenstein, A., Lakhotia, A., Parida, L.: Malware phylogeny generation using permutations of code. J. Comput. Virol. **1**(1–2), 13–23 (2005)
8. Gartner Inc: http://www.gartner.com/press_releases/asset_129199_11.html (2005)
9. Johannes, F.: A study using n-gram features for text categorization. Technical Report OEFAI-TR-9830, Austrian Institute for Artificial Intelligence (1998)
10. Kephart, J.O., Sorkin, G.B., Arnold, W.C., Chess, D.M., Tesauro, G.J., White, S.R.: Biologically inspired defenses against computer viruses. In: Proceedings of the 14th IJCAI, pp. 985–996, Montreal (1995)
11. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: KDD '04: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 470–478. ACM Press, New York (2004)
12. Lefevre, E., Colot, O., Vannoorenberghe, P.: Belief function combination and conflict management. Inf. Fusion **3**(2), 149–162 (2002)
13. McGraw, G., Morrisett, G.: Attacking malicious code: a report to the infosec research council. IEEE Soft. **17**(5), 33–41 (2000)
14. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
15. Murphy, C.K.: Combining belief functions when evidence conflicts. Decis. Support Syst. **29**(1), 1–9 (2000)
16. Nachenberg, C.: Understanding and managing polymorphic viruses. Technical Report, The Symantec Exterprise Papers: Vol. XXX
17. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976)
18. Schultz, M.G., Eskin, E., Zadok, E., Bhattacharyya, M., Stolfo, S.J.: Mef: Malicious email filter – a unix mail filter that detects malicious windows executables. In: Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 245–252. USENIX Association, Berkeley (2001)
19. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy, p. 38. IEEE Computer Society, Washington (2001)
20. Sentz, K.: Combination of evidence in Dempster–Shafer theory. Ph.D. Thesis, SNL, LANL, and Systems Science and Industrial Engineering Department, Binghamton University
21. Smets, P.: Belief functions: The disjunctive rule of combination and the generalized bayesian theorem. Int. J. Approx. Reason. **9**(1), 1–35 (1993)
22. Szor, P.: The Art of Computer Virus Research and Defense. Addison Wesley, Reading (2005)
23. Ting, K.M., Witten, I.H.: Issues in stacked generalization. J. Artif. Intell. Res. **10**, 271–289 (1999)
24. Vx heavens: http://www.vx.netlux.org
25. Witten, I., Frank, E.: Data mining: Practical machine learning tools and techniques with Java implementations. Morgan Kaufmann, San Francisco (2000)
26. Wolpert, D.H.: Stacked generalization. Technical Report LA-UR-90-3460, Los Alamos (1990)
27. Yang, Y., Pedersen, J.O.: A comparative study on feature selection in text categorization. In: Fisher, D.H. (ed.) Proceedings of ICML-97, 14th International Conference on Machine Learning, Nashville, pp. 412–420. Morgan Kaufmann, San Francisco (1997)
28. Yoo, I., Ultes-Nitsche, U.: Non-signature based virus detection: Towards establishing unknown virus detection technique using som. J. Comput. Virol. **2**(3) (2006)
29. Zhang, B., Srihari, S.N.: Class-wise multi-classifier combination based on dempster-shafer theory. In: Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision (2002)