VILO: A Rapid Learning Nearest-Neighbor Classifier for Malware Triage

Arun Lakhotia · **Andrew Walenstein** · **Craig Miles** · **Anshuman Singh**

Abstract VILO is a lazy learner system designed for malware classification and triage. It implements a nearest neighbor (NN) algorithm with similarities computed over Term Frequency × Inverse Document Frequency (TFIDF) weighted opcode mnemonic permutation features (N-perms). Being an NN-classifier, VILO makes minimal structural assumptions about class boundaries, and thus is well suited for the constantly changing malware population. This paper presents an extensive study of application of VILO in malware analysis. Our experiments demonstrate that (a) VILO is a rapid learner of malware families, i.e., VILO's learning curve stabilizes at high accuracies quickly (training on less than 20 variants per family is sufficient); (b) similarity scores derived from TDIDF weighted features should primarily be treated as ordinal measurements; and (c) VILO with N-perm feature vectors outperforms traditional N-gram feature vectors when used to classify real-world malware into their respective families.

Keywords malware · triage · VILO · familial classification · antivirus

This research work was sponsored in part by funds from Air Force Research Lab and DARPA (FA8750-10-C-0171) and from Air Force Office of Scientific Research (FA9550-09-1-0715).

Arun Lakhotia
Center for Advanced Computer Studies
University of Louisiana at Lafayette
E-mail: arun@louisiana.edu

Andrew Walenstein
School of Computing and Informatics
University of Louisiana at Lafayette
E-mail: walenste@ieee.org

Craig Miles
Center for Advanced Computer Studies
University of Louisiana at Lafayette
E-mail: craig@craigmil.es

Anshuman Singh
Center for Advanced Computer Studies
University of Louisiana at Lafayette
E-mail: asingh@louisiana.edu

1 Introduction

New malware are frequently related to previous malware through evolution. In order to combat detection by anti-virus scanners, malware frequently evolves by way of rapid modify-and-release cycles, creating numerous variants from a common origin. This commonality defines a 'family' to which related variants belong. Consider the data from "Microsoft Security Intelligence Report Volume 8: July through December 2009" [33]; there were 126,204,254 variants of malware found within the second half of 2009. Clearly, the multitudinous nature of the observed malware indicates that it cannot all have been built ex novo. Instead, the vast majority are merely modified previous versions. These modifications are generally indicative of polymorphism, i.e., malware that changes its decryption or unpacking code for each variant, or metamorphism, i.e., malware for which all variants are functionally equivalent, but their internal structures differ [5, 49, 6]. These two techniques allow for rapid creation of an essentially unlimited number of variants. According to Microsoft's data, only a few hundred families are observed in the wild in a typical half-year period. For a malware analyst, the ability to quickly identify the family to which a malicious executable belongs is a boon.

Classification techniques from machine learning and statistics have often been used to classify malware, such as viruses and worms [41,1,24,29,36]. Two different types of malware classification are possible: binary and familial. In the binary malware classification problem, an unknown executable is classified as either being malicious or benign. Conversely, in the familial malware classification problem, a known-to-be malicious executable is classified as belonging to a particular group of malware. Familial classification can be used in malware triage—labeling and prioritizing suspicious executables for further analysis. While it has been shown that no detector for a particular family of malware without false positives can exist [4,10], and by extension, no perfectly accurate familial classifier can either, past works in familial classification [47,49,17] have shown promising results.

In this paper we describe VILO, a malware classification system that is suitable for familial classification of malware, and thus triage. VILO makes use of three components for effective familial classification: N-perm feature vectors, Term Frequency × Inverse Document Frequency (TFIDF) weighting of features [18], and the nearest neighbor search algorithm. N-perms are a variation of N-grams—a commonly used feature in malware classification obtained by sliding a window of size n over program objects (bytes, opcodes, etc.). N-perms are identical to N-grams except that the order of characters within the N-perm is irrelevant for matching purposes. They are robust against some code obfuscations, such as instruction reordering [8]. TFIDF weighting of features ensures that features that are common across many types of executables are not overly emphasized. Nearest neighbor search is simple and effective as it does not require construction of a classification model while also making minimal structural assumptions about the separability of classes of data. Retraining involves simply adding a new variant to the database, as opposed to the explicit model building required by other classifiers like decision trees and support vector machines.

N-perms were introduced in our previous work [19] in the context of clustering for phylogeny generation. In [47], it was shown that usage of N-perm feature vectors provided greater accuracy in the malware classification scenario than did N-gram feature vectors when applied to an artificially constructed data set. However up until now, no results have been presented that describe the efficacy of either type of VILO feature vector when used to classify actual in-the-wild malware. In this paper, we present results of the first such study and we show that usage of N-perms provides greater accuracy than does usage of N-grams for real-world malware classification.

TFIDF weighting has previously been used in malware classification for byte N-grams [24] and opcode N-grams [36]. We propose its application for N-perms in the familial classification scenario. We also show empirically that for any nearest neighbor algorithm, TFIDF weighted similarity scores depend on the relative sample sizes of the classes represented in the training database. This implies that similarity scores should be treated as ordinal measurements.

We evaluated VILO's efficacy in familial classification and studied the per-family training set size necessary to obtain high accuracy. We found that VILO can achieve high familial classification accuracy when trained on few variants from each family (less than 20). Since VILO uses nearest neighbor search, a small training database implies quick query resolution. This ability to learn rapidly makes VILO suitable for production environments. Further gains in classification accuracy are attainable in exchange for greater memory consumption and query resolution time.

The contributions of this paper are:

- It describes the VILO method for familial malware classification, and shows that VILO is a quick and efficacious learner of real-world malware.
- It presents an evaluation which suggests that usage of N-perm feature vectors results in between 0.14% and 5.42% fewer misclassifications than does usage of N-gram feature vectors when used in the context of familial malware classification.
- It shows that TFIDF weighted similarity scores should be treated as ordinal measurements whenever possible.
- It presents an evaluation which indicates that usage of the VILO system is not suitable for binary classification of malware.

The rest of this paper is organized as follows: In Section 2, we give some background and discuss related work. In Section 3, we discuss some of the challenges in classifying executables that arise when using machine learning methods for the purpose of malware classification. The VILO system for malware classification is described in Section 4. Our malware and benign data sets are described in Section 5. Section 6 presents discussion and evaluations of the VILO system's efficacy when used in the familial malware classification scenario, as well as comparison against other methodologies. Discussion and evaluations pertaining to the usage of the VILO system in the binary classification of maliciousness scenario are presented in Section 7. The conclusions drawn from the present work are discussed in Section 8.

2 Background and Related Work

The methods used for malware classification can be categorized into expert-based and learning-based. Expert-based methods rely on domain experience of an AV analyst to determine rules for malware classification. Learning-based methods, on the other hand, automatically learn the rules from examples. Learning-based methods can be categorized into two types: supervised and non-supervised. Supervised methods train a classifier on labeled examples whereas unsupervised methods use unlabeled examples and group them into clusters using some measure of similarity. VILO is based on supervised learning, where a collection of labeled malicious executables is used for training.

The construction of learning-based classifiers involves multiple stages. First, properties of data are extracted into feature vectors. This is followed by selection of the most relevant features. Then, a learning algorithm is used to learn from training data (labeled examples). Finally, the performance of the classifier obtained after learning is evaluated using methods like cross-validation.

The use of the "right" kind of features is critical to the success of learning-based methods. Masud et al. [28] describe a variety of features for worm detection, intrusion detection, botnet detection, and malicious executable detection. One of the earliest works on learning-based methods for malware detection, by Tesauro et al. [44], used byte tri-grams as features. In an influential paper by Schultz et al. [42], three types of features were proposed: byte sequences (N-grams), printable strings, and DLL calls. In both the works ([44] and [42]), features were boolean, i.e., they represented presence or absence in the executable. In a departure from the use of boolean features, Kolter et al. [23,24] proposed using continuous weights of byte sequences called TFIDF. TFIDF has been used widely in text categorization [43] and is defined as $TF \times IDF$. TF or term frequency is the number of occurrences of a particular term i (TC_i) in a document (e.g., an executable) normalized by the total number of terms ($|D|$) in the document ($TF_i = TC_i/|D|$). *Document frequency* (DF_i) is the number of documents containing the term. *Inverse document frequency* is defined as $IDF = |D|/DF_i$, or $IDF = \log_{10}(|D|/DF_i)$ if log scaling is desired. VILO uses the definition of IDF without the log scaling:

Wang et al. [48] were the first to propose the use of N-grams of instruction opcodes (obtained from disassembling the executable) as features. Later, Karim et al. [19] used instruction mnemonics to construct N-perms—instruction reordering resistant N-grams. Moskovitch et al. [36] did an extensive study on opcode based malware detection. They show classifiers using opcode N-grams perform better than classifiers using byte N-grams. VILO can use either N-perms and N-grams of instruction opcodes with TFIDF weights.

Most of the previous works on learning-based methods have been concerned with binary classification, in which a given executable of unknown class is classified as malicious or benign. There is another classification problem, called *familial classification*, where a malicious executable of unknown family is classified as belonging to a known malware family. Familial classification is used in triage for prioritizing sample queues [37]. It can also be used for sample examination and observation with static and behavioral features, respectively. Familial classification can be used for attribute discovery of a sample during the examination phase of automated filtering. The results in this paper show VILO to be a more effective method for familial classification than it is for binary classification.

Towards familial classification of malware, Stamp et al. [49,46,40] have achieved success using Hidden Markov Models (HMMs) to learn about and detect variants of a metamorphic virus, i.e., a virus for which all variants are functionally equivalent, but their internal structures differ. An HMM models a memoryless stochastic process in which the states themselves cannot be directly observed, but rather only external observations can be made from which it may be possible to infer with some probability the current state. Using opcode sequences extracted from related viruses as observations, Stamp et al. trained the HMM to best fit the observed sequences. After learning the HMM, they were able to use it to determine the likelihood that other executables belonged to the same virus family. Another project in which familial classification was performed is BitShred [17]. BitShred is a feature-type agnostic approach to malware triage classification that uses feature hashing and a probabilistic variation on traditional Jaccard similarity. Using byte 16-grams as features, BitShred attained high accuracy in familial classification of malware. These two approaches are revisited in Subsection 6.4.

3 Issues in Classifying Executables

The application of machine learning methods to the problem of classifying malware is not straightforward. There are domain-specific issues that must be accounted for when applying machine learning methods to the purpose of classifying malware in particular. Some of these issues are discussed below:

1. **Change Sensitivity**: A good executable classifier should ensure that minor variations between the ex-

ecutables do not significantly throw off the reported match. Thus, the similarity function used to compare executables must be insensitive to such minor variations. For instance, the similarity function should be insensitive to the differences in branch targets due to instruction reordering or insertion. If a malware author adds statements to the beginning of her program, then all of the subsequent branch targets will move. Any algorithm or function that is not overly sensitive to such changes is more suitable. Simple reorderings of modules, procedures, or even individual instructions should similarly be accounted for. The need for change sensitivity is most clearly expressed in metamorphic malware, of which Lin and Stamp [27] provide a good discussion of the metamorphic techniques employed by malware authors. Many of the common differencing or similarity functions for strings are not suitable in this regard, because they focus specifically on such orderings. Sequencing, after all, is the defining characteristic of a string. Examples of such matching techniques include longest common subsequence, Levenshtein or edit distances, and Hamming distances. Techniques such as these have previously been applied to find matching source code [21], however they are not suitable for the purposes of executable matching due to the specific insensitivities required.
2. **Handling common code**: Many even unrelated executables often share some commonalities, such as standard function prologues and epilogues. The match method should account for these commonalities so that emphasis is not unduly placed on features that are relatively insignificant with respect to identifying matches.
3. **Efficiency**: Efficiency is always an issue. Although, higher level abstractions of programs like control flow graphs, program slices, or semantic-level representations could be more useful, they are computationally more expensive to extract. Certainly various similarity measures can be defined over such higher-level representations, such as control flow [12, 11, 3, 25, 7, 22]. However, besides being potentially more costly and difficult to scale to thousands of program comparisons, these methods are more vulnerable to obfuscations [26]. As such, it is important to explore what can be done with the near-minimum abstraction of the extracted information from a program.

These are some of the issues that the VILO method, described in the next section, addresses.

4 VILO Method for Malware Classification

In this section, we present and describe a malware classification system called VILO (derived from VIrus phyLOgeny). VILO, at its core, is an executable similarity measurement engine upon which we have implemented a 1-nearest neighbor classifier. The VILO system, N-grams, and N-perms, etc., have all also previously been discussed in our group's prior work [19, 47].

4.1 VILO Overview

The VILO system is an adaptation of text retrieval methods that use TFIDF term-vector query matching. These methods have previously been used for matching text documents to queries, and for the related task of detecting duplicate documents [50, 15].

This subsection overviews the techniques that are employed by VILO in order to ensure that the approach meets the previously identified challenges in matching executables (sensitivity to change, handling common code, and efficiency). The details of the topics overviewed are covered more thoroughly in the subsequent subsections. These techniques can be summarized as follows:

1. Whole programs are compared, meaning the set of features that are compared are relatively comprehensive. The features we use, "N-perms", are extracted from abstracted disassemblies and are not very sensitive to minor changes introduced in malware variants. This can be contrasted to traditional signature based techniques which look for a highly focused but diagnostic feature, such as a unique byte sequence.
2. A vector model is used for comparison. Feature counts are converted into vectors that can be compared by measuring their cosine similarity. This is fast and simple to calculate.
3. Feature weights are calculated from the corpus of executables on which VILO has been trained. The weighting scheme addresses the identified concern of handling common code effectively.

4.2 VILO Specifics

4.2.1 Feature types: N-gram and N-perms

Text-based comparison methods frequently use "N-grams". An N-gram is a sequence of n characters found in succession within some document. Various different definitions of "characters" can be used: it could be letters, words, sentences, or paragraphs, and some could

STRING	2-GRAMS
The cat is in.	th he ec ca at ti is si in
Is the cat in?	is st th he ec ca at ti in

Fig. 1: 2-grams on ASCII text

be modified (e.g., made lower case) or filtered entirely (e.g., filtering spaces). Figure 1 shows an example using 2-grams on text, while mapping to lower case and filtering out spaces and punctuation. From the list of 2-grams, the differences and commonalities are tallied. The differences are {si, st} and the commonalities are {at, ca, ec, he, in, is, th, ti}; both of which are then counted to arrive at a similarity score. N-grams are simple to extract (using a sliding n-sized window), and are easy to compare. However, by their very nature, such character sequences may be overly sensitive to local sequencing, especially for larger values of n.

It is possible to reduce the importance of sequence information. This is accomplished by using "N-perms", which we first described in Karim et al. [19] and which were also described independently by Wong et al. [49]. N-perms are exactly like N-grams except that the ordering of the characters is not considered during the matching. For any sequence of n characters that can be taken to be an N-gram, an N-perm represents every possible permutation of that sequence. For example, the possible 3-grams of *abcab* are *abc, bca, cab*, each with only one occurrence. However, for the same string, there is only one 3-perm, *abc*, which has three occurrences. Statement reorderings tend to have less of an effect when using N-perm feature vectors. Usage of N-perms in favor of N-gram feature vectors also provides the benefit of reducing the feature vector space, as was seen by the need to store only the single N-perm feature *abc* in the above example.

There are many ways to apply N-grams and N-perms to programs. The most commonly reported technique is to use raw bytes as characters [24,1,20]. However, extracted embedded strings, disassembled instructions, or some combination of all of these could be used.

For our VILO system, we have chosen to use abstracted assembly as characters. The *abstracted assembly* of a given instruction is simply its mnemonic. For example, the abstracted assembly of the x86 assembly language instruction *mov eax, 5* is *mov*. Figure 2 further illustrates the usage of abstracted assembly for the generation of N-perms and N-grams. In prior work, we concluded that usage of 5-grams and 10-perms is a suitable choice for malware classification when using abstracted assembly as features [19], and we will use those values for n in our subsequent analyses.

4.2.2 Feature weights: TFIDF

For each executable VILO will use in the classification process, a feature vector is generated. An executable's feature vector contains all of its unique features, either as N-perms or N-grams, and each of those features' associated number of occurrences within the executable.

Not all features are equally useful for classification purposes. Certain sequences of instructions are common to many Windows Portable Executable (PE) [34] files, such as function prologues and epilogues. These features should not be heavily considered when attempting to find matches, as they commonly belong to even unrelated executables. Conversely, a feature which only occurs in a small subset of executables is more likely to be useful for classification. It is not desirable to manually specify these weights by, for example, requiring an expert to create a list of common operation sequences to be filtered (i.e., a "stoplist").

In order to set the weight of each feature automatically, we employ a Term Frequency × Inverse Document Frequency (TFIDF) weighting scheme [18]. In this scheme, a feature is weighted by the inverse of how frequently it appears in the set of executables. In the case of text retrieval, this weighting encodes the heuristic logic that a match on the relatively rare word "constitution", say, will tend to be more meaningful than matches on the more common word "the." For example, given the feature sets for two viruses $v_1 = [3\ 4\ 2\ 1]$ and $v_2 = [4\ 5\ 1\ 0]$, using standard cosine similarity,

$$\begin{aligned} sim(v_1, v_2) &= \frac{v_1 \cdot v_2}{|v_1||v_2|} \\ &= \frac{3 \times 4 + 4 \times 5 + 2 \times 1 + 1 \times 0}{\sqrt{3^2 + 4^2 + 2^2 + 1^2}\sqrt{4^2 + 5^2 + 1^2 + 0^2}} \\ &= 0.957 \end{aligned}$$

If there are 10 programs in the database and the four features' document frequencies are 9, 8, 3, and 2 respectively, then weighted versions of the vectors are $w_1 = [3/9\ 4/8\ 2/3\ 1/2] = [.33\ .25\ .66\ .50]$ and $w_2 = [4/9\ 5/8\ 1/3\ 0/2] = [.44\ .63\ .33\ .00]$, and the similarity becomes $\text{sim}(w_1,\ w_2) = 0.795$. The lowered similarity score reflects the reduced importance of the features that occur in many files.

Smart application of feature weighting enables VILO to quickly learn which sequences of opcode mnemonics are most indicative of a particular family of malware, and conversely, which sequences are irrelevant to the matching process. Indeed, our empirical results indicate that through usage of TFIDF weighting, VILO is able to learn about malware families quickly. Because feature weights are updated each time a new variant of

```
55                   push  ebp
b8 11 00 00 00       mov   $0x11,eax
89 e5                mov   esp,ebp
57                   push  edi
99                   cltd
56                   push  esi
c7 45 e4 11 00 00 00 mov   $0x11,0xffffffe4(ebp)
```

(a) disassembly

FEATURE TYPE	FEATURES EXTRACTED
2-grams	`push_mov, mov_mov, mov_push, push_cltd, ...`
2-perms	`push_mov, mov_mov, push_cltd`

(b) features extracted from operations (boxed column)

Fig. 2: Feature extraction from abstracted disassembly

a malware family is added to the training data, VILO's understanding of the features which serve to identify that malware family evolve along with the family itself.

4.2.3 Classification algorithm: Nearest-neighbor

VILO uses the nearest-neighbor algorithm, which is essentially a lazy learning method [14]. Lazy learners store all the training data and delay the classification until a query is given to the classifier. For an unknown executable U, VILO finds U's most similar matches from its training database of existing feature vectors. It iteratively calculates the pairwise similarity between U and every feature vector in the database, then sorts the result in descending order of similarity. The final result is a ranked list and the classification we subsequently assign to U is that of the executable at the head of that list, i.e. the nearest neighbor. The algorithm is $O(NM)$ where N is the number of programs in the database and M is the vector length. The advantage of using a lazy learning method is that VILO need not build a new classification model each time a new variant is added to the training database. Since new variants of malware appear in the wild frequently, the training database may be updated to improve the accuracy without the overhead of building a new model. Another advantage is that nearest neighbor classifiers are low bias classifiers with minimal structural assumptions about classes in data. Our results indicate effectiveness of this algorithm in familial classification of malware.

4.3 Implementation

VILO employs the linear sweep disassembler objdump[1] to obtain the abstracted assembly of entire executables; all bytes, including those in both the code and data sections, are disassembled. We selected objdump in favor of more sophisticated disassemblers, such as IDA Pro[2] or that built into OllyDbg[3], because we are primarily concerned with obtaining as many representative features as possible, including instructions disassembled from non-code data. Disassembling all of the bytes of a binary is much easier to accomplish with objdump than it is with the other disassemblers. Furthermore, the ability to match homogeneous features drawn from both code and data likely outweighs the marginal benefit obtained from more accurate disassembly.

A feature vector generator and a VILO search server are implemented in C. The feature vector generator takes as input the modified objdump's abstracted assembly output, and generates either N-perm or N-gram feature vectors. The VILO search server then reads these feature vectors into a database and calculates the feature weightings. It then listens for queries, which consist of other feature vectors to be matched against. The returned result for a query is a ranked list comprised of both the matches found and their respective weighted cosine similarities.

[1] `http://www.gnu.org/software/binutils/`
[2] `http://www.hex-rays.com/products/ida/index.shtml`
[3] `http://www.ollydbg.de/`

5 Data Set

Our data set consists of malicious executables belonging to the following four families: Backdoor.Win32.Hupigon, Backdoor.Win32.PcClient, Rootkit.Win32.Agent, and Virus.Win32.Parite. The malware was acquired from an AV vendor who established the ground truth, i.e., determined to which of those families each of the malicious executables belong. The ground truth was determined by human analysts conducting manual analysis, and thus we consider it quite reliable.

Backdoor.Win32.Hupigon is the backdoor component of a greater family of malware called Win32.Hupigon. It is registered as a system service and opens a backdoor server that allows other computers to connect to and control the infected computer in various ways [30]. Backdoor.Win32.PcClient is a backdoor trojan family whose variants contain several malicious components, including a key logger, backdoor, and a rootkit [32]. Rootkit.Win32.Agent installs a rootkit on the infected system. A rootkit is software that provides an attacker with continued administrator level access to the infected computer while actively hiding its own presence. Virus.Win32.Parite is a polymorphic file infecting virus that infects all Windows portable executable files found on local and shared drives [31].

We have attempted to ensure that the binaries are not packed or compressed by using the packer identification tool PEiD. We generally assume that the familial identification of each malicious executable as determined by the antivirus vendor is accurate. However, we recognize that a mislabeled sample within a small training set may significantly skew the results. Therefore, our experiments are designed to be resilient against such inaccuracies.

In Section 7, we also use a set of benign executables which we gathered from the system directories of several fresh installations of Microsoft Windows.

6 Familial Classification of Malware

The ability to effectively classify a malicious executable into its family is particularly important when conducting malware related triage. The first concern in a triage scenario is to identify the intent and capability of the aggressor, therefore quick identification of the type of malware used by an attacker is of the utmost importance. In this section, we describe and subsequently employ our evaluation method to determine and compare the effectiveness of both N-perm and N-gram VILO feature vectors for the purpose of classifying malicious executables into their respective families.

6.1 Effect of Familial Training Set Size

In order to evaluate and compare the effectiveness of N-perm and N-gram VILO feature vectors, several evaluation parameters must be defined. Such parameters include the values of N for the N-perm and N-gram VILO feature vectors (for which we have previously selected 10 and 5 respectively [19]). Another such evaluation parameter is the per-family training set size to be used. This subsection presents an experiment we conducted that shows the effect of incrementally increasing the per-family training set size on the average classification accuracies observed when using either N-perm or N-gram VILO feature vectors.

6.1.1 Experimental Design

It should be recognized that the evaluation of malware classifiers presents domain specific issues that must be accounted for in the selection of an evaluation methodology. For example, it is common in classifier evaluations to train on larger amounts of data than what may be needed. The often-seen k-fold cross-validation evaluation technique makes use of a large training set and a relatively smaller verification set. However, it may be unnecessary and unreasonable to only evaluate a malware classifier when it is trained on a very large sample from each malware family. For a malware classification system such as VILO, each time a new unknown executable is to be classified, its similarity with every executable in the training set must be calculated; obviously a costly computation. As such, it is important for malware classifiers such as ours to be accurate even if they are only allowed to be trained on small samples for some or all of the known malware families. The present experiment determines how quickly the VILO system, using either type of feature vectors, is able to learn about the families in its training set.

6.1.2 Procedure

We randomly selected from each of our four malware families 100 malicious executables, which we further partitioned into training and verification sets comprised of 40 and 60 malicious executables from each malware family, respectively. Both N-perm and N-gram VILO feature vectors were generated for each malicious executable. Then, for both types of feature vectors, we employed the following algorithm:

```
for $j = 1 \rightarrow 40$ (the size of the training sets) do
    From the training set of each malware family, select
    the first $j$ feature vectors and train VILO on them.
    for each feature vector $V$ in the combined verification set with samples from each malware family do
        Make VILO classify $V$.
        Record if $V$'s classification is correct or not.
    end for
    Calculate and record the percentage of correct classifications for the current value of $j$.
end for
```

The entire experiment was run twenty times with randomly selected samples, and the percentages of correct classifications for each value of j were averaged across all of the twenty runs. The averaged percentages of correct classifications represent the average classification accuracy when VILO is trained with j samples from each family, and averaging over multiple experimental runs helps to ensure that the results of training on a mislabeled malicious executable do not significantly skew our findings. Finally, we plot the recorded average classification accuracies against the size of the familial training sets in order to construct a learning curve [9].

6.1.3 Results

The learning curves generated by our experiment for both N-perm and N-gram VILO feature vectors are shown in Figure 3. The per-family training sample size is on the x-axis and the accuracy obtained is on the y-axis. When the per-family training sample size was 5, the average accuracy when using N-grams was 70.17% and for N-perms it was 76.29%; for samples of size 10, the average accuracies were 76.38% and 81.94% respectively; for samples of size 20, the average accuracies were 80.04% and 84.75% respectively; and for samples of size 40, the average accuracies were 83.65% and 86.44% respectively.

6.1.4 Discussion

We observe that usage of N-perm VILO feature vectors provides greater accuracy than N-gram VILO feature vectors for each size of the training set. Additionally, we note that the classifier appears to reach near-optimal accuracy quickly; the accuracy climbs dramatically as the training set size initially increases before quickly leveling off. These results show, for the first time, that usage of N-perm VILO feature vectors is preferable over usage of N-gram VILO feature vectors when attempting to classify real-world malicious executables. Because one impetus behind the creation of N-perm VILO feature vectors was to make malware classification resilient against code-reordering obfuscations, we further infer that such obfuscation techniques may indeed be used by malware authors to evade detection and identification.

Learning curves for each of the four individual malware families are in the Appendix. The curves for the four individual families look generally similar to that of the combined learning curve in Figure 3. One notable exception is that of Backdoor.Win32.Hupigon, whose N-perm curve peaks at 92% when the training size is 4. As the training size increases beyond 4, the accuracy gradually falls (by less than a fraction of 1% on average). This indicates that overtraining may cause a decrease in accuracy for some malware families.

6.2 Matched Pairs T-Test for a Specified Training Set Size

Having shown graphically the differences in accuracy between usage of N-perm and N-gram VILO feature vectors for familial malware classification, we now present and employ a method for statistically quantifying the difference in accuracy for a specified per-family training set size.

6.2.1 Experimental Design

In order to select the size of the sample from each family on which to train the VILO database in our subsequent analyses, we recognize two factors. First, we must again concern ourselves with keeping the size of the training set as small as possible in order to reduce the time and space costs of performing malware classifications. Second, we have shown via the previous experiment that the overall increase in accuracy as the per-family training set size increases is subject to the law of diminishing returns. We therefore strive to select a value which is not prohibitively large but that also provides a non-negligible increase in accuracy over a training set size of one less from each family.

We arbitrarily choose that an average increase in accuracy, provided by incrementation of the per-family training set beyond some fixed size, of less than 0.5% is to be considered negligible. We find that this occurs for N-perms when the training set size per family is 10 and for N-grams when the training set size is 11. As such, we select a per-family training set size of 11 on which to conduct further analyses.

An experimental set was constructed, consisting of 191 randomly selected malicious executables sampled from each of our four malware families. From each familial set of 191 malicious executables, 180 malicious executables were randomly selected to be members of

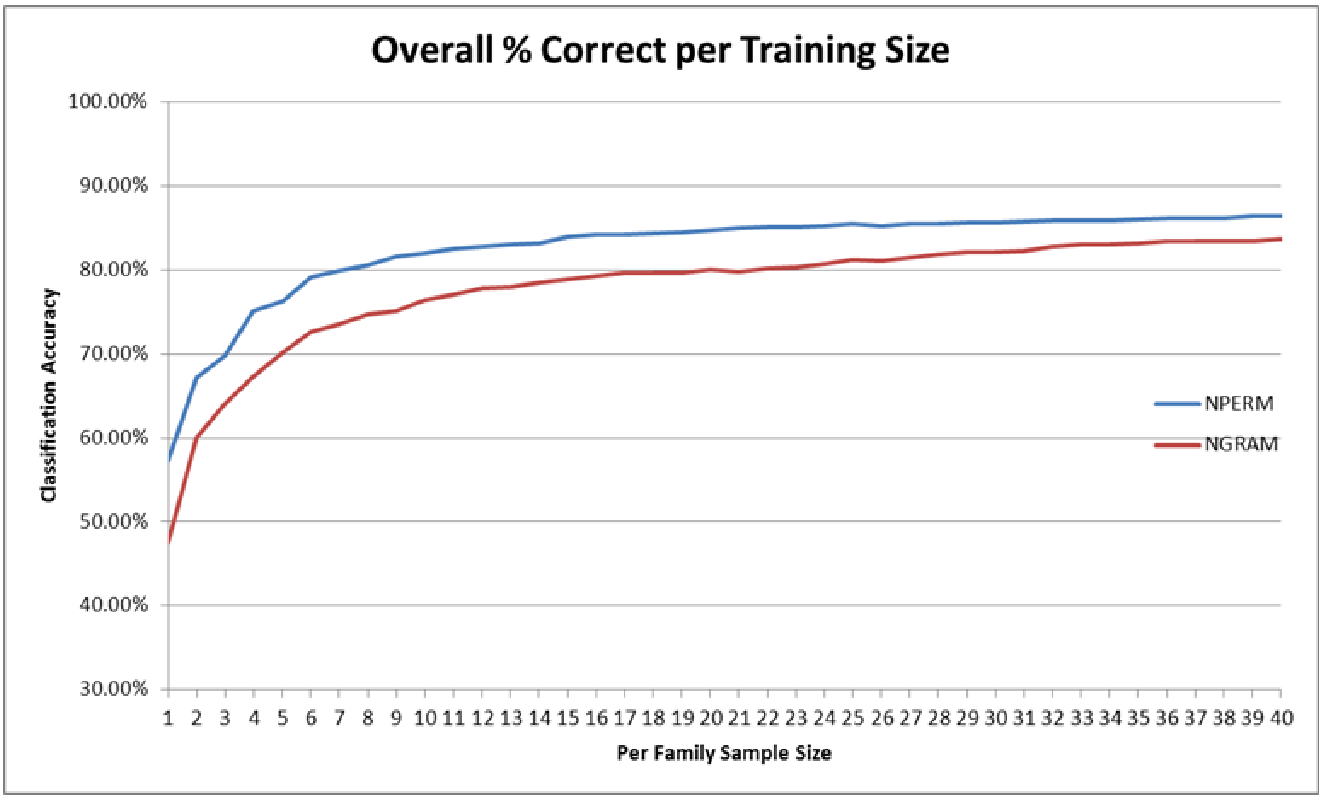

Fig. 3: Combined Learning Curve

the verification set. Thus, the size of the whole verification set was 720. The remaining 11 malicious executables from each family were joined to construct the training set with total size 44.

6.2.2 Procedure

Let T and V be the aforementioned training and verification sets respectively. Generate N-perm and N-gram VILO feature vectors for the samples in both T and V, which results in N-perm sets T_P and V_P, and N-gram sets T_G and V_G. For the N-perm feature vector sets, train VILO on T_P, then $\forall v \in V_P$, determine the nearest neighbor vector. If the family of the nearest neighbor vector matches that of v, then the result is said to be a correct classification, otherwise a misclassification has occurred. Similarly, for the N-gram feature vector sets, train VILO on T_G, then $\forall v \in V_G$, determine the nearest neighbor vector and record the classifications results in the same manner as before.

For each malicious executable in the verification set and for both types of feature vectors derived from those executables (V_P and V_G), we record whether or not a misclassification occurred. A misclassification is encoded as a 1 and a correct classification as a 0. Furthermore, we calculate the difference of the N-perm and N-gram misclassification encodings. To illustrate, see Table 1 for some sample results. The first column gives the unique identifier of the particular malicious executable under consideration. The second and third columns indicate whether or not that malicious executable was correctly classified using N-perm or N-gram VILO feature vectors respectively. The fourth column is the difference of the second and third columns. From Table 1, we see that the malicious executable with identifier Parite_5 was misclassified when N-gram feature vectors were used, but when using N-perm feature vectors it was correctly classified as belonging to the Parite malware family.

Because each observation for an N-perm feature vector has a matched observation in the results for an N-gram feature vector, the table consists of dependent matched pairs. This type of data lends itself to being analyzed using a paired T-test on the differences [16].

6.2.3 Results

When the per-family training set size threshold S at which negligible increase in accuracy was obtained was chosen to be 11, the mean difference (mean = -0.0278, standard deviation = 0.3605, sample size = 720) was significantly less than zero, t(719)= -2.0675, two-tail

file_id	nperm_misclassified	ngram_misclassified	difference
Hupigon_8	0	0	0
Parite_5	0	1	-1
PcClient_2	1	1	0

Table 1: Extract of Misclassification Results.

p = 0.0390, providing evidence that usage of N-perm VILO feature vectors provides greater accuracy over usage of N-gram VILO feature vectors. A 95% confidence interval about mean misclassification difference is (-0.0014, -0.05415). This means that we can be 95% confident that usage of N-perm VILO feature vectors results in between 0.14% and 5.42% fewer misclassifications than does usage of N-gram VILO feature vectors when the samples are drawn from the four malware families described above.

6.2.4 Discussion

For the four malware families available to us, our statistical analysis reveals that usage of N-perm VILO feature vectors for familial malware classification is significantly more accurate than is usage of N-gram VILO feature vectors. As such, we further conclude that using N-perm VILO feature vectors may be favorable to N-gram VILO feature vectors for the purpose of familial malware classification over the universe of malware. Of course, this hypothesis may be further tested by running similar experiments on samples from a greater number of malware families.

For real world usage of the VILO system, the per-family training set size to be used must be carefully selected by observing the trade-off incurred when increasing classification accuracy at the costs of slowing performance and requiring more space.

As with most systems for which many parameters must be chosen ahead of time, it is a very difficult task to choose the best values for VILO's parameters. We acknowledge that our chosen values for n in N-perms and N-grams and our selected training set size may not be the best in all cases. Conducting more analyses to determine best values for these parameters provides a good basis for future work.

6.3 Evaluation of Familial Classification using K-fold Cross Validation Estimates

We recognize that our approach for evaluating a malware classifier for familial classification differs from that generally seen in the literature. Works which evaluate the performance of a malware classifier often make use of the k-fold cross-validation technique (e.g. [23, 45]). For the reasons previously detailed, k-fold cross-validation does not necessarily lend itself well to classifier evaluations where it is desirable to have a smaller training set than verification set. Hence, we constructed a tertiary experiment which makes use of the 10-fold cross validation technique while retaining our ability to limit the number of files on which the classifier is allowed to be trained.

6.3.1 Experimental Design

In k-fold cross-validation (see Rodriguez et al. [39] for a good primer on the technique), the data set S is randomly partitioned into k folds (subsets) of equal size. For each fold, one partition is used as verification or the test set and the remaining partitions are used as the training set. The classifier is trained and tested for each fold until all partitions have been used as test set. The accuracies obtained for each fold are averaged to obtain the cross validation estimate.

6.3.2 Procedure

As in Subsection 6.2.1, we use 11 as the learning curve threshold at which accuracy stabilizes. From each of our four malware families, we gathered random samples of size 1,000. Each family sample was subsequently broken into 10 disjoint folds of 100 variants, then one fold was selected to be the verification set while the remaining nine folds were joined into a training set of size 900. 11 malware were randomly selected from these 900 on which to train VILO. We conducted ten such runs such that each fold was used as the verification set exactly once. For each run, we generated N-gram VILO feature vectors and N-perm VILO feature vectors for all of the executables in the verification and training sets. Then we trained VILO on the training set, verified with the verification set, and finally we recorded the results.

6.3.3 Results

Combined across all runs, usage of N-perm VILO feature vectors resulted in 3,286 correct classifications out of a total possible 4,000, whereas usage of N-gram VILO feature vectors resulted in 3,236 correct classifications.

The resulting 10-fold cross-validation prediction error estimations are thus $(4000 - 3286)/4000 = 0.178$ and $(4000 - 3236)/4000 = 0.191$, respectively.

6.3.4 Discussion

Once again, we have shown that usage of N-perm VILO feature vectors provides greater accuracy in familial classification of malware than does usage of N-grams. Additionally, the difference in classification accuracy observed with the usage of the 10-fold cross-validation technique falls well within the confidence interval generated by our previous analyses, thereby serving to further confirm that result.

6.4 Comparison to Other Systems

An obvious system to compare VILO against would be commercial anti-virus software, as has been done in related work (e.g. [49]). However, two factors hinder comparison between VILO and commercial AV.

First, commercial AV vendors are more concerned with detecting whether an executable is malicious than with associating the correct familial label to it. As such, these commercial AV products often classify malware into generic 'bucket families', e.g., Avast's Win32:Trojan-gen, Avira's TR/Downloader.Gen, BitDefender's Trojan.Generic, and ClamAV's Trojan.Dropper. Indeed, malware analysts have told us that we should put little faith in any family labels given by their product [35]. See Section 2.2 of Bailey et al. [2] for further discussion on this topic.

The second hindrance is in setting up a fair comparison. The purpose of malware triage classifiers like VILO is to identify the family of an unknown malicious executable; if its family could be determined by the analyst's AV software, then the need to employ a more complex system such as VILO would be obviated. Assuming this VILO use-case, the AV product should only have signatures as of the moment the analyst would have received the new file. Since at that moment the commercial AV product can have no signature for the new malicious executable (otherwise it would not be unknown to the analyst), it would not be capable of detecting it. Thus, if the AV products had the signature they would have during real world usage of VILO, then VILO would perform much better than them; on the other hand, comparing VILO against AV products with more up-to-date signatures is a poor choice as it violates the VILO use-case.

We instead compare VILO against other machine learning based familial triage classifiers found in related works. In Wong et al. [49], a Hidden Markov Model over disassembled opcode observations was learned and used to detect viruses created in-lab with NGVCK[4]. They reported success detecting viruses from that particular family (perfect true positive vs. false negative accuracy in detecting NGVCK viruses), however other malware was also classified as belonging to the NGVCK family (false positives). Jang et al. [17] described and evaluated a system called BitShred for malware triage. Their data set consisted of 102,391 malware with family labels (i.e., ground truth) determined by the ClamAV anti-virus product. Using byte 16-grams to find a malicious executable's five nearest-neighbors, they reported 94.2% precision and 92.2% recall. While BitShred's results suggest familial classification accuracy greater than our own, at least one of the families BitShred classified malware into was a generic bucket family, Trojan.Dropper. We suspect that BitShred's familial classification accuracy would be more similar to VILO's if their ground truth did not include bucket families and was determined in a more precise fashion than from application of a single commercial anti-virus product; in contrast, ground truth for our data set was established via manual analysis by trained malware analysts.

We have shown that VILO, when used for familial malware classification, can attain an accuracy of 86.44% while training on relatively few variants per family. Such accuracy is competitive with other approaches [49, 17], though due to differences in data sets, methods for determining ground truth, and evaluatory methodologies, direct comparison of reported accuracies is difficult.

7 Binary Classification Using VILO

Binary classification is the process of classifying an unknown executable as either being malicious or benign. Thresholding is employed in order to perform binary classification with VILO: if the similarity of an executable with respect to its nearest neighbor is below a certain threshold, it is likely not known malware. Usage of VILO for binary classification gives rise to two issues:

- *Threshold selection*: Since VILO uses TFIDF, for which weights vary depending on the familial sample sizes in the training database, the selection of a particular threshold depends on the current composition of the training set.
- *Abstaining classification*: If an executable's nearest-neighbor similarity lies below a certain threshold,

[4] NGVCK (Next Generation Virus Creation Kit) is a metamorphic virus generator that outputs syntactically different, semantically equivalent x86 ASM source code for viruses.

then we can say that it is not similar to anything in the training database. However, we cannot say that it is benign, as the training database itself may just be lacking. Hence, saying "I don't know" is a safer approach than definitively declaring such an executable benign. Classifiers like these have been called abstaining classifiers in machine learning literature [38].

The following two subsections discuss these issues in more detail, then in Subsection 7.3 we evaluate VILO's efficacy in performing binary classification of unknown executables when trained on real-world malware. The summary result of the evaluation is that VILO is not suited for binary classification.

7.1 Dependence of Threshold on Training Database Contents

The goal of using TFIDF weighting in the VILO system is to reduce the importance of features that appear commonly in many vectors while giving more consideration to features that are specific to smaller subsets of the vectors in the VILO database. A possible concern in this approach is that when the VILO database is largely populated with related vectors, if another related vector is matched against the VILO database, then the TFIDF weighting may reduce the similarity score of the nearest neighbor vector to a significant degree. The end result of such a reduced nearest neighbor similarity score may be that the closest match returned, while correct, may appear to be relatively dissimilar. We illustrate the dilemma with a quick inline experiment.

We hypothesize that a VILO database populated with many related vectors will result in a lower nearest neighbor similarity score when matching against another related vector than will a VILO database populated with fewer related vectors.

N-perm VILO feature vectors were generated from 100 unique variants belonging to the malware family Backdoor.Win32.Hupigon. Additionally, N-perm VILO feature vectors were generated from 100 known-to-be benign Windows executables. Two more Hupigon variants were selected and N-perm VILO feature vectors for them were generated to be used in verification against the VILO database trained with vectors from the larger sets. For both of the verification vectors, the following steps were performed:

```
for $i = 1 \rightarrow 100$ do
    Train VILO on i malware vectors and (100 - i) benign vectors.
    Compute the similarity score of the two Hupigon samples with corresponding nearest neighbor.
end for
```

Figure 4 illustrates the results of implementing the described process. In both cases, as the percentage of related malware vectors in the VILO database increases, the similarity score of the nearest neighbor vector when matching against another related malware vector decreases. The result is as we intuitively expect TFIDF weighting to behave.

These graphs show that due to the use of TFIDF weighting, as the percentage of a malware family as a part of the whole trained VILO database increases, the similarity scores returned by all members of that family decrease. It follows then that it is not possible to fix a universal similarity score threshold that a nearest neighbor vector must meet in order to be considered a true match. If such a threshold is to be used, it must depend on the size of the largest familial training set, and must therefore be determined uniquely for each running of the classifier.

The results of this experiment indicate that a given similarity score computed by the VILO system is not necessarily indicative of either a 'good' or 'bad' match. Rather, it is generally better to treat such weighted similarity scores as ordinal measurements whenever possible. In cases where the actual similarity scores must be treated as interval scale measurements, such as in the binary classification of malicious versus benign, care must always be taken to select an appropriate similarity score threshold with regard to the current contents of the VILO database.

7.2 Benign as a Class

Some prior evaluations of binary classifiers have included benign executables in the training data [41, 24, 29]. However, we think benign executables should not be used for training and the binary classification should be viewed as an abstaining classification problem [38]. This is because most machine learning based classifiers, such as VILO, learn about classes in a positive manner, i.e., the existence of previously seen malicious features in some unknown executable indicates that it is likely malicious as well. If the match of a query to the nearest-neighbor in the training data is below a certain similarity threshold, then it is safer to abstain and say "I don't know" than declaring it benign.

If benign executables must be matched against, then that matching should take place via comparison with a white-list, i.e. an expert-selected set of known-to-be benign executables against which a perfect match, and only a perfect match, indicates benignness with much confidence.

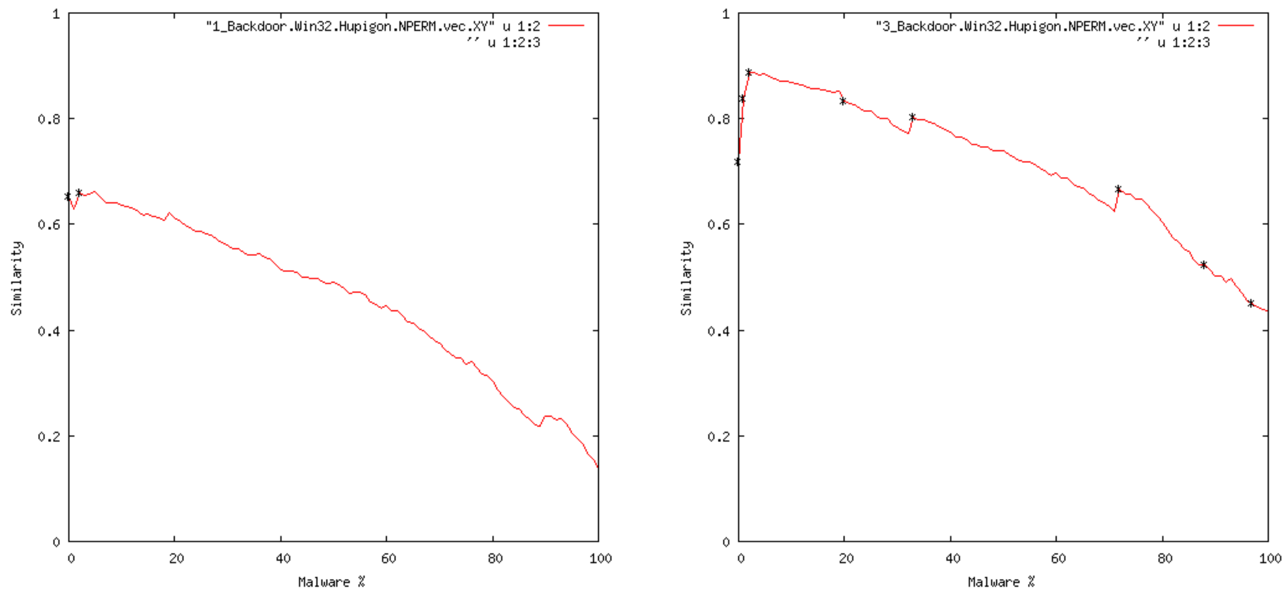

Fig. 4: TFIDF Effect on Nearest Neighbor Similarity Score

7.3 Evaluation of Binary Classification with VILO

For classifiers that generate a similarity score or probability value for a given input, a standard technique to evaluate the classifier is to generate the Receiver Operating Characteristic (ROC) curve [13]. For a set of generated classification results where ground truth is known, an ROC curve plots the true positive rate against the false positive rate as the required similarity or probability threshold for a positive classification is incrementally increased from 0 to 1.

We generated ROC curves for VILO's binary classification using both N-perm and N-gram feature vectors. Samples of 800 executables were taken from each of our four malware families; 600 of which were added to the training set and the remaining 200 were added to the testing set. The testing set was extended by adding 200 benign executables to it. The ROC curves generated using this data are shown in Figure 5.

As is evident from the presented ROC curves, neither usage of N-perm nor N-gram feature vectors provided desirable results in the binary classification scenario. For both types of feature vectors, the true positive rate only surpasses 50% when the false positive rate is greater than 60%. As such, we conclude that VILO is not suitable for use in binary classification.

8 Conclusions

VILO is a nearest-neighbor algorithm based system for malware triage that uses TFIDF weighted features from unpacked disassembled binaries. VILO is effective for familial classification and it learns about malware families quickly, i.e., it attains high accuracy while only training on relatively few variants from each malware family. On the other hand, VILO is not as satisfactory in binary classification of maliciousness. Our evaluations using statistical tests and k-fold cross-validation show VILO performs better with N-perms compared to N-grams, which suggests that N-perms may provide a performance boost in many systems that employ traditional N-grams. Additionally, we showed that similarity scores calculated over weighted feature vectors should be treated as ordinal measurements whenever possible.

Appendix

Learning curves derived from usage of both N-perm and N-gram VILO feature vectors for Backdoor.Win32.Hupigon, Backdoor.Win32.PcClient, Rootkit.Win32.Agent, and Virus.Win32.Parite are shown herein (Figs 6, 7, 8, 9).

Acknowledgements The authors are grateful for Prof. Mihai Giurcanu's help in identifying proper statistical evaluation methods. Furthermore, we wish to thank Suresh Golconda, Chris Parich, Michael Venable, Matthew Hayes, and Christopher Thompson for their past work, without which this paper would not have been possible.

References

1. T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code.

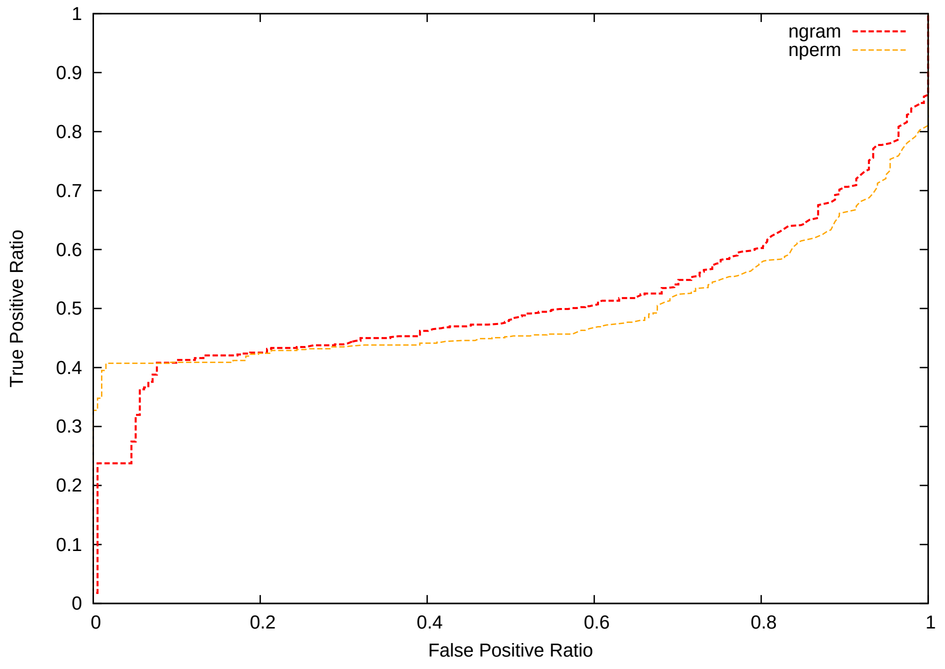

Fig. 5: ROC Curves for Binary Classification of Malware using N-perms and N-grams

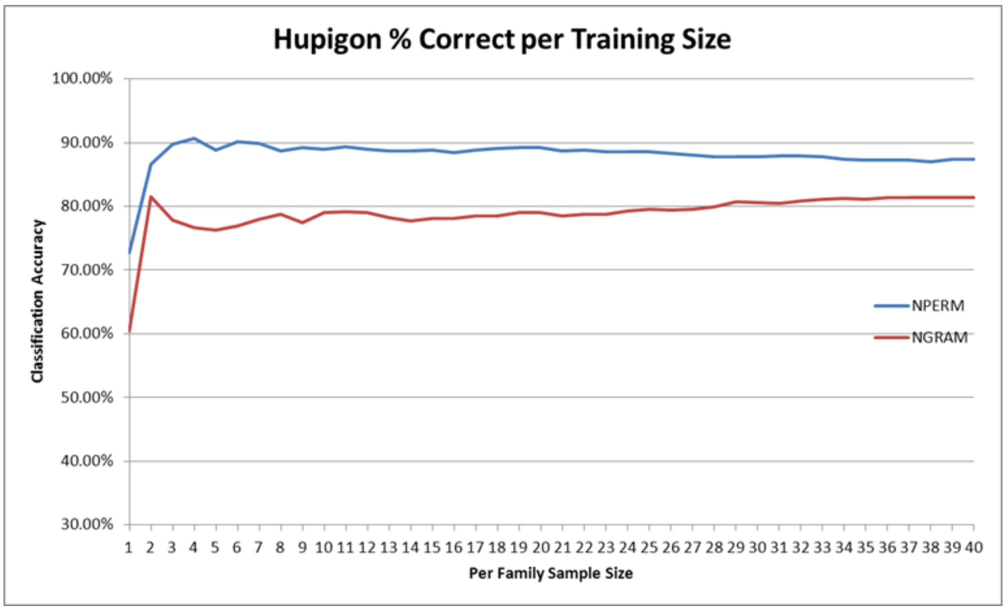

Fig. 6: Backdoor.Win32.Hupigon Learning Curves

In *Proc. of the 28th Annual International Computer Software and Applications Conference, 2004 (COMP-SAC'04)*, volume 2, pages 41–42. IEEE, 2004.
2. M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, RAID'07, pages 178–197, Ber, Heidelberg, 2007. Springer-Verlag.
3. E. Carrera and G. Erdélyi. Digital genome mapping–advanced binary malware analysis. In *Virus Bulletin Conference*, pages 187–197, 2004.
4. D. Chess and S. White. An undetectable computer virus. In *Proceedings of Virus Bulletin Conference*, volume 5, 2000.
5. M. Chouchane and A. Lakhotia. Using engine signature to detect metamorphic malware. In *Proceedings of the 4th ACM workshop on Recurring malcode*, pages 73–78. ACM, 2006.
6. M. Chouchane, A. Walenstein, and A. Lakhotia. Statistical signatures for fast filtering of instruction-substituting metamorphic malware. In *Proceedings of the 2007 ACM*

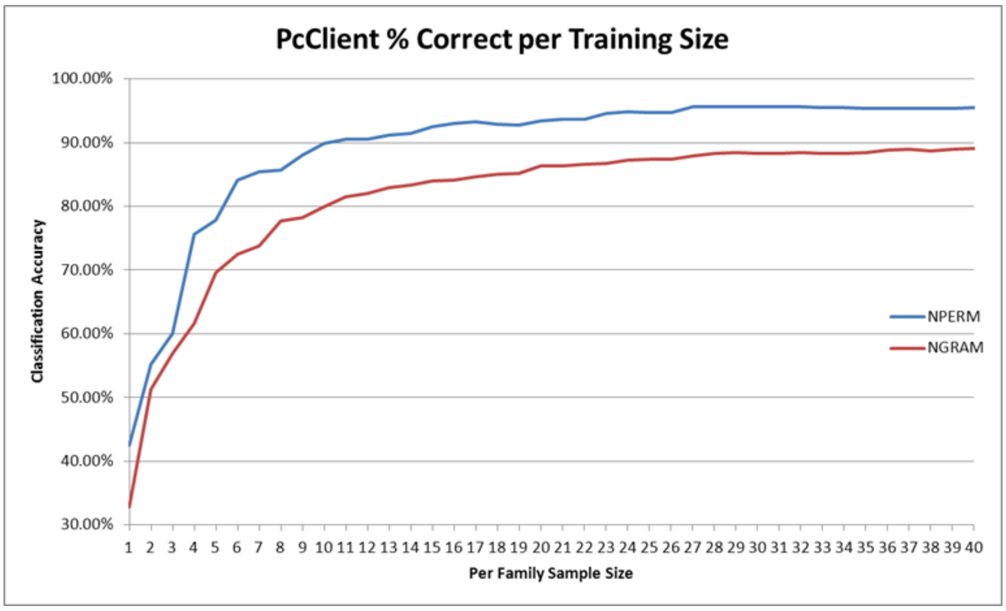

Fig. 7: Backdoor.Win32.PcClient Learning Curves

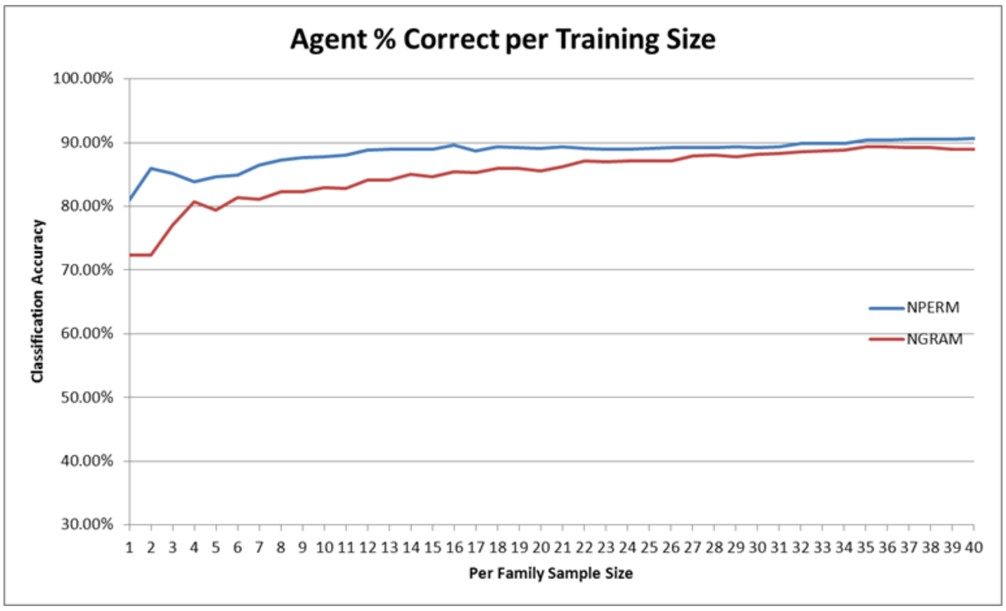

Fig. 8: Rootkit.Win32.Agent Learning Curves

workshop on Recurring malcode, pages 31–37. ACM, 2007.

7. M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. In *Security and Privacy, 2005 IEEE Symposium on*, pages 32–46. IEEE, 2005.
8. F. Cohen. Operating system protection through program evolution. *Computers & Security*, 12(6):565–584, 1993.
9. R. Duda, P. Hart, and D. Stork. *Pattern classification*, volume 2. Wiley New York:, 2001.
10. E. Filiol and S. Josse. A statistical model for undecidable viral detection. *Journal in Computer Virology*, 3(2):65–74, 2007.
11. H. Flake. More fun with graphs. In *Proceedings of Black-Hat Federal*, 2003.
12. H. Flake. Structural comparison of executable objects. In *Proc. of the International GI Workshop on Detection of Intrusions and Malware & Vulnerability Assessment, number P-46 in Lecture Notes in Informatics (DIMVA'04)*, pages 161–174, 2004.
13. D. Green and J. Swets. *Signal detection theory and psychophysics*, volume 1974. Wiley, New York, 1966.
14. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, second edition, 2006.
15. T. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American So-*

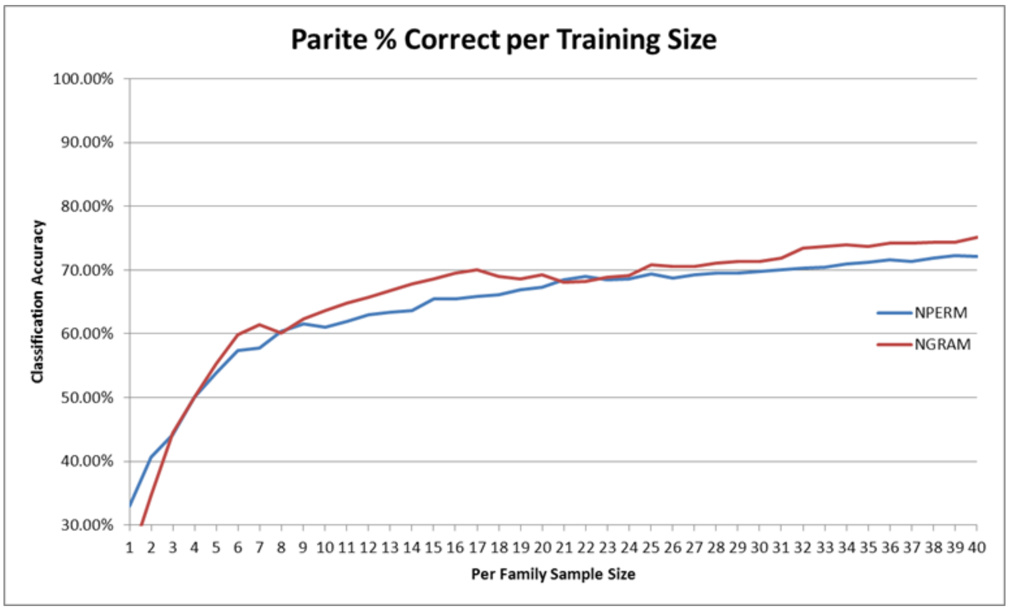

Fig. 9: Virus.Win32.Parite Learning Curves

ciety for Information Science and Technology, 54(3):203–215, 2003.

16. R. Hogg, J. McKean, and A. Craig. Introduction to mathematical statistics, 2005.
17. J. Jang, D. Brumley, and S. Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 309–320. ACM, 2011.
18. K. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.
19. M. Karim, A. Walenstein, A. Lakhotia, and L. Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1):13–23, 2005.
20. J. Kephart and W. Arnold. Automatic extraction of computer virus signatures. In *4th Virus Bulletin International Conference*, pages 178–184, 1994.
21. M. Kim and D. Notkin. Program element matching for multi-version program analyses. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 58–64, 2006.
22. J. Kinable and O. Kostakis. Malware classification based on call graph clustering. *Journal in computer virology*, 7(4):233–245, 2011.
23. J. Kolter and M. Maloof. Learning to detect malicious executables in the wild. In *Proc. of the tenth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 470–478, 2004.
24. J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2721–2744, 2006.
25. C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *Recent Advances in Intrusion Detection*, pages 207–226. Springer, 2006.
26. A. Lakhotia and P. Singh. Challenges in getting formal with viruses. *Virus Bulletin*, 9(1):14–18, 2003.
27. D. Lin and M. Stamp. Hunting for undetectable metamorphic viruses. *Journal in computer virology*, 7(3):201–214, 2011.
28. M. Masud, L. Khan, and B. Thuraisingham. *Data Mining Tools for Malware Detection.* CRC Press, 2011.
29. M. M. Masud, L. Khan, and B. Thuraisingham. A hybrid model to detect malicious executables. In *Proc. of the IEEE Intl. Conf. on Communications (ICC 2007)*, pages 1443–1448, 2007.
30. Microsoft. Microsoft Malware Protection Center Backdoor:Win32/Hupigon. `http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Backdoor%3AWin32%2FHupigon`, March 2006.
31. Microsoft. Microsoft Malware Protection Center Virus:Win32/Parite.b. `http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?name=Virus%3aWin32%2fParite.B`, April 2007.
32. Microsoft. Microsoft Malware Protection Center Backdoor:Win32/PcClient. `http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Backdoor%3AWin32%2FPcClient`, Mar. 2009.
33. Microsoft. Microsoft security intelligence report July through December 2009. `http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=11722`, May 2010.
34. Microsoft. Microsoft PE and COFF Specification. `http://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx`, October 2011.
35. C. Miles and A. Lakhotia. Personal correspondance with malware analysts. personal communication, 2012.
36. R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici. Unknown malcode detection using opcode representation. In *European Conference on Intelligence and Security Informatics 2008 (EuroISI08), Lectures Notes in Computer Sciences Volume 5376*, pages 204–215. Springer, 2008.
37. I. Muttik. Malware mining. In *VB2011: Proceedings of 21st Virus Bulletin Conference*, 2011.

38. T. Pietraszek. On the use of roc analysis for the optimization of abstaining classifiers. *Machine Learning*, 68(2):137–169, 2007.
39. J. Rodriguez, A. Perez, and J. Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):569–575, 2010.
40. N. Runwal, R. Low, and M. Stamp. Opcode graph similarity and metamorphic detection. *Journal in Computer Virology*, pages 1–16, 2012.
41. M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proc. of S&P 2001: IEEE Symposium on Security and Privacy*, pages 38–49, 2001.
42. M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proc. of S&P 2001: the IEEE Symposium on Security and Privacy*, pages 38–49, 2001.
43. F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys*, 34(1):1–47, 2002.
44. G. Tesauro, J. Kephart, and G. Sorkin. Neural networks for computer virus recognition. *IEEE Expert*, 11(4):5–6, 1996.
45. R. Tian, L. Batten, and S. Versteeg. Function length as a tool for malware classification. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 69–76, 2008.
46. A. Toderici and M. Stamp. Chi-squared distance and metamorphic virus detection. *Journal in Computer Virology*, pages 1–14, 2012.
47. A. Walenstein, M. Venable, M. Hayes, C. Thompson, and A. Lakhotia. Exploiting similarity between variants to defeat malware. *Proceedings of BlackHat Briefings DC 2007*, 2007.
48. J. H. Wang, P. S. Deng, Y. S. Fan, L. J. Jaw, and Y. C. Liu. Virus detection using data mining techniques. In *Proc. of the 37th Intl. Carnahan Conf. on Security Techology*, pages 71–77, 2003.
49. W. Wong and M. Stamp. Hunting for metamorphic engines. *Journal in Computer Virology*, 2:211–229, 2006.
50. J. Zobel and A. Moffat. Exploring the similarity space. *ACM SIGIR Forum*, 32(1):18–34, 1998.