Exploring Discriminatory Features for Automated Malware Classification

Guanhua Yan[1], Nathan Brown[2], and Deguang Kong[3]

[1] Information Sciences (CCS-3)
Los Alamos National Laboratory
[2] Department of Electrical and Computer Engineering
Naval Postgraduate School
[3] Department of Computer Science
University of Texas, Arlington

Abstract. The ever-growing malware threat in the cyber space calls for techniques that are more effective than widely deployed signature-based detection systems and more scalable than manual reverse engineering by forensic experts. To counter large volumes of malware variants, machine learning techniques have been applied recently for automated malware classification. Despite the successes made from these efforts, we still lack a basic understanding of some key issues, such as what features we should use and which classifiers perform well on malware data. Against this backdrop, the goal of this work is to explore discriminatory features for automated malware classification. We conduct a systematic study on the discriminative power of various types of features extracted from malware programs, and experiment with different combinations of feature selection algorithms and classifiers. Our results not only offer insights into what features most distinguish malware families, but also shed light on how to develop scalable techniques for automated malware classification in practice.

1 Introduction

The sheer volume of malware has posed serious threats to the health of cyber space. According to Symantec, as many as 286 million unique malware variants have been witnessed in 2010 alone [35]. It is thus impossible for us to manually reverse engineering every malware variant and study their malicious behaviors. Fortunately, many of these malware variants share similar origins. According to the 2006 Microsoft Security Intelligence report [20], more than 75 percent of malware variants detected can be categorized into as few as 25 families. If we grasp the trend of how each of these malware families evolves, we are at an advantageous position of developing effective, yet efficient, techniques to mitigate the tremendous malware threats.

Studying evolution of distinct malware families calls for methods that can quickly classify a large number of malware variants into their corresponding lineages. Major AV (Anti-Virus) companies commonly use signature-based approaches, which are known to be error-prone. On the other hand, manually reverse engineering malware to find their lineages requires advanced skills and is thus too time consuming to keep up with the current pace of ever-evolving malware programs. To overcome these challenges, we investigate machine learning techniques that learn *automatically* from samples labeled

K. Rieck, P. Stewin, and J.-P. Seifert (Eds.): DIMVA 2013, LNCS 7967, pp. 41–61, 2013.

either by malware forensic experts or from consensus among major AV companies to classify new malware variants.

Such supervised learning, however, demands discriminatory information that is representative of malware lineages. In parlance of machine learning, this is an issue of *feature selection*. The motivations behind feature selection for automated malware classification are manifold. First, feature selection can relieve us from collecting unnecessary features, some of which may be difficult to extract from malware programs. Second, features that are found capable of accurately classifying malware families offer insights into the key differences among malware families. This objective renders feature selection more desirable than those methods relying on dimension reduction, which projects features into a space with little semantic meaning. Last but not least, performances of many classification algorithms hinge on the number of features used.

Against this backdrop, the goal of this work is to explore discriminatory features for automated malware classification. We extract various types of features from malware programs, and for each type of these features, we study its discriminative power as well as how to select the most useful ones for automated malware classification. As performances of feature selection and classification techniques heavily depend on application domains, we consider various combinations of feature selection and classification methods to gain deep insights into what algorithms perform well for automated malware classification.

Our major observations from this comprehensive study are summarized as follows. **(1)** Different types of features vary significantly in their abilities in classifying malware families. Our study has shown that features extracted from PE headers possess high discriminative power in classifying malware families. This observation is encouraging as the cost of extracting features from PE headers of executable programs is low compared against other types of features such as those from dynamic traces. **(2)** For the same type of features, we find that a small number of features are usually sufficient for a classifier to reach its peak classification performance. This further confirms the importance of feature selection, and suggests that an automated malware classification system could rely on only a selected set of malware features to improve its scalability. **(3)** Among the four classifiers we have tested in this study, we find that a variant of the decision tree classifier (i.e., C4.5) performs consistently well in classifying all malware families. The decision tree classifier is known to have scalability advantages over other classifiers such as SVM and kNN [16], which offers hope for developing fast and scalable tools in classifying a large number of malware variants.

2 Related Work

Malware feature extraction is a key step towards malware classification/clustering analysis. Previously, many types of malware features have been used to classify or cluster malware instances, such as byte sequence n-gram [31,12,25], instructions in execution traces [1], PE headers [34,26], function call graphs [8], control flow graphs [15], and system calls [11,4,6]. Our study compares the discriminative power of different feature types, which has not been thoroughly treated previously.

Since the initial works of Schultz *et al.* [31] and Kolter *et al.* [12], machine learning techniques have been used in many efforts to automatically classify unknown malware

files into different categories [25,28]. SVM, kNN and the decision tree are several most popular classifiers used for malware classification problems [27,21]. Similar to our efforts in this study, Ye *et al.* compared the performances of SVM, the decision tree, and Naive Bayes in detecting whether a program is malicious or not based on the API calls made by executable programs, and found that Naive Bayes performs the worst and the decision tree performs slightly better than SVM [38]. Our work differs from theirs in that we consider the problem of classifying malware into different families, and the types of features in this work are far more diverse than what they have studied.

In contrast to malware classification that requires labeled samples for training, malware clustering automatically identifies multiple classes of malware that share similar features in an unsupervised learning fashion [3,4,10]. Although feature selection is performed for the purpose of malware classification in this work, some methodologies used here can be applied for malware clustering as well, although feature selection for clustering is a much harder problem due to absence of class labels [30].

3 Dataset Description

Malware Dataset. We use a dataset submitted to Offensive Computing [22] in Febuary 2011. It contains 526,179 unique malware variants collected in the wild. Using the `pefile` utility [24] and the `PEiD` signature database (uploaded date: Feb 10, 2011) detects that 30% of them are packed. Among all the malware variants detected to be packed, the distribution of the top ten packers is shown in Figure 1. Armadillo and UPX are the two most popular packers used to pack malware in the malware dataset.

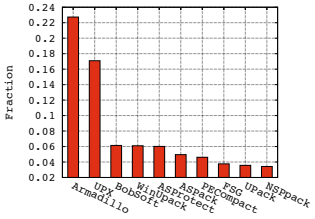


Fig. 1. Top 10 packers

AV Software	Result	Family name
McAfee	Vundo.gen.m	Vundo
NOD32	a variant of Win32/Adware.Virtumonde.NBG	Virtumonde
Kaspersky	Trojan.Win32.Monderb.gen	Monderb
Microsoft	Trojan:Win32/Vundo.BY	Vundo
Symantec	Packed.Generic.180	GENERIC

Fig. 2. Classification results of a malware instance by five AV software

We upload all our malware variants to the VirusTotal website [36], and find that the 43 AV software vary in their capabilities in detecting malware variants in the malware dataset. The top-performed software is AntiVir, which is able to detect almost 80% of the malware variants; by contrast, ByteHero has detected only 4.8% of them. Among the 43 software, the mean detection rate is 60.5% and the standard deviation is 18.2%.

To obtain labeled data, we take the following steps to choose malware variants for which we are confident in their families.

Step 1: Family name identification. From the VirusTotal output, we note that the naming scheme of each AV software differs significantly. For instance, some of the classification results of the malware with md5 `bd264800202108f870d58b466a1ed315`

are shown in Figure 2. To identify the family name from the detection result, our algorithm partitions the result into a list of words based on the set of separators that are used by the AV software. Next, our algorithm removes from the list those words that are too generic to indicate a malware family, such as "Win32," "gen," "Trojan," as well as numbers. The first word on the list is returned as the malware family name, or "GENERIC" is returned if the list becomes empty.

Step 2: Alias resolution. Another challenge is that different AV software use different family names for the same malware variant [19]. Following the same example, both McAfee and Microsoft classify it as a variant of the `Vundo` family, and NOD32 detects it as one of the `Virtumonde` family. Kaspersky detects it as one of the `Monderb` family, which is part of the bigger `Monder` family, and Symantec detects it as a generic packed malware variant from which we cannot identify the malware family name.

To resolve aliases named differently by AV software, we start from a few well-known malware family names, such as Bagle and Bifrose, and identify the AV software that use these family names. We select those malware variants that are commonly classified as these family names by these software, and check how another AV software classifies the selected malware variants. If the majority of the malware variants are classified as a specific family name, we obtain the alias of this malware family used by that AV software. Due to the large variation in detection results by different AV software, we consider only those from McAfee, Kaspersky, Microsoft, ESET (NOD32), and Symantec. But the methodology developed in this study can be easily extended to incorporate detection results from other AV software. For these five AV software, we resolve the aliases for a few well-known malware families as shown in Table 1.

Table 1. Alias resolution and malware selection

Family	McAfee	Symantec	Microsoft	Kaspersky	NOD32	Full	Unpacked
Bagle	Bagle	Beagle	Bagle	Bagle	Bagle	285	152
Bifrose	Backdoor-CEP	Bifrose	Bifrose	Bifrose	Bifrose	2085	1677
Hupigon	BackDoor-AWQ	Graybird	Hupigon	Hupigon	Hupigon	11001	4748
Koobface	Koobface	Koobface	Koobface	Koobface	Koobface	439	371
Ldpinch	PWS-Ldpinch	Ldpinch	Ldpinch	Ldpinch	Ldpinch	310	190
Lmir	PWS-Legmir	Lemir	Lemir	Lemir	Lmir	366	181
Rbot	Sdbot	Spybot	Rbot	Rbot	Rbot	2565	923
Sdbot	Sdbot	Sdbot	Sdbot	Sdbot	Sdbot	629	253
Swizzor	Swizzor	Lop	Swizzor	Swizzor	Swizzor	1826	1276
Vundo	Vundo	Vundo	Vundo	Monder	Virtumonde	3278	2853
Zbot	Zbot/PWS-Zbot	Zbot	Zbot	Zbot	Zbot	1317	1233
Zlob	Puper	Zlob	Zlob	Zlob	Zlob	2747	2146

Step 3: Malware selection by majority agreement. Finally, we select a subset of malware variants from the malware dataset for which their malware families can be established with high confidence. To this end, for each malware variant in the malware dataset, we check the malware family in Table 1 into which each of the five AV software classifies it. If *four* of them classify it into the same family, we select this malware variant as belonging to this family. In total, we have selected 26,848 malware variants belonging to 12 malware families, as shown in the right part of Table 1. We also show the number of unpacked malware variants among those selected in each malware family.

Benign Executable Dataset. In addition to the malware dataset, we also have collected a set of 597 benign executable programs. Similarly, we use the `pefile` utility and

the PEiD signature database to detect the packer information. The breakdown of packers detected to pack benign executables is as follows: Armadillo (7.7%), InstallShield (2.5%), UPX (1.7%), PECompact (0.5%), WinZip (0.34%), ASPack (0.17%), and Wise Installer Stub (0.17%). Overall, the fraction of packers detected from benign executables (13%) is lower than that from the malware dataset (30%).

Balanced Datasets. From Table 1, we note that the number of samples varies significantly among different malware families. Let the *imbalance ratio* be the ratio of the number of samples in the family with the most instances to that in the family with the least instances. The full dataset, which contains both packed and unpacked instances, has an imbalance ratio of 38.6, and the dataset that contains only unpacked samples has an imbalance ratio of 31.2. As imbalanced data pose severe challenges to learning and classification [7], we use the simple down-sampling technique to create a dataset which contains the same number of instances from each malware family and benign executables. More specifically, in the *balanced* dataset, we randomly choose 150 unpacked instances from each family, and randomly choose 150 benign executables. By contrast, the *imbalanced* dataset contains all instances that are detected to be unpacked by PEiD. The reason that we ignore packed instances is that the packing procedure of a malware program is not representative of its true functionality, *and* different malware families can use the same packer, which further complicates automated malware classification.

4 Methodology

To study the discriminative power of different types of features, we consider four widely used classifiers: *Naive Bayes*, *kNN*, *SVM*, and the *decision tree* (we use the C4.5 decision tree in this study). In parlance of machine learning, the performance of a classifier can be quantified with *precision*, *recall*, and *F-1*. Let the number of true positives, false positives, true negatives, and false negatives be n_{tp}, n_{fp}, n_{tn} and n_{fn}, respectively, when we use a classifier c. Then, the precision metric is defined as $n_{tp}/(n_{tp} + n_{fp})$, and the recall metric is $n_{tp}/(n_{tp} + n_{fn})$. The F-1 metric is the harmonic mean of precision and recall, that is, $2n_{tp}/(2n_{tp} + n_{fp} + n_{fn})$. An ideal classifier would have F-1 metric close to 1, implying that both precision and recall are close to 1.

Feature selection algorithms fall into three different categories. *Filter methods* rank features independently and choose features with highest scores for classification. Since features chosen by a filter method are blind to the classifier used later, they may not perform the best for that specific classifier. This distinguishes filter methods from *wrapper methods*, which aim to choose a subset of features that perform the best under a specific classifier. A wrapper method is usually much slower than a filter method, as for each candidate subset of features, it has to use a specific classifier to evaluate the classification performance. *Embedded methods* are another type of feature selection techniques, which exploit sparsity by forcing weights associated with non-chosen features to be zero. Due to execution performance concern, we do not consider wrapper methods in this study. In the following, we introduce three filter methods, ReliefF, Chi-squared (or χ^2) and F-statistics, and two embedded methods which are L1-regularized methods.

ReliefF [14]. The ReliefF score of a feature is calculated as follows. Randomly choose m reference instances $\{x_i\}_{i=1,2,\ldots,m}$, and for each reference instance x_i, let set H_i

contain its k closest samples in the same class and set M_i its k closest samples in a different class. The ReliefF score is:

$$R = \frac{\sum_{i=1}^{m} \sum_{y \in M_i} |x_i - y|}{\sum_{i=1}^{m} \sum_{y \in H_i} |x_i - y|}, \tag{1}$$

where $|x_i - y|$ denotes the distance between x_i and y. We choose $m = 100$ and $k = 20$.

Chi-squared [18]. This method evaluates an individual feature with respect to the classes. Numerical features are discretized into intervals. The χ^2 score of a feature is:

$$C = \sum_{i=1}^{m} \sum_{k=1}^{K} \frac{(A_{ik} - E_{ik})^2}{E_{ik}}, \tag{2}$$

where m is the number of intervals, K the number of classes, A_{ik} the number of instances of class k in the i-th interval, R_i the number of instances in the i-th interval, S_k the number of instances in class k, N the total number of instances, and $E_{ik} = R_i \times S_k / N$.

F-Statistics [13]. The F-statistic score of a feature f is calculated as follows:

$$F = \frac{\sum_{k=1}^{K} \frac{n_k}{K-1} (\mu_k - \mu)^2}{\frac{1}{n-K} \sum_{k=1}^{K} (n_k - 1) \sigma_k^2}, \tag{3}$$

where K is the total number of classes, μ is the mean of all instances on feature f, and for any class $k : 1 \leq k \leq K$, n_k is the number of instances in class k, and μ_k and σ_k are the mean and standard deviation of instances in class k on feature f, respectively.

L1-Regularized Methods [39]. Consider two-class labeled data $\{\mathbf{x_i}, y_i\}_{i=1}^{l}$, $\mathbf{x_i} \in R^n$, and $y_i \in \{1, -1\}$. Under the L1-regularized logistic regression model (**L1-logreg**),

$$\min_{\mathbf{w}} P^{LR}(\mathbf{w}) = C \sum_{i=1}^{l} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}}) + ||\mathbf{w}||_1, \tag{4}$$

and under the L1-regularized linear SVM model (**L1-SVC**),

$$\min_{\mathbf{w}} P^{SVM}(\mathbf{w}) = C \sum_{i=1}^{l} \max(1 - y_i \mathbf{w}^T \mathbf{x_i}, 0) + ||\mathbf{w}||_1, \tag{5}$$

where X^T is the transpose of X, and $||X||_1$ is the L1-norm of X. Since the L1-norm of the $||\mathbf{w}||$ is included in the objective functions, L1-regularized methods force sparsity in weights in $\mathbf{w}$. Only features with non-zero weights are chosen for classification. Parameter C controls the number of non-zero features indirectly. When the number of features chosen is predefined, we use a binary search method to find the right C that gives the exact number of non-zero weights we want.

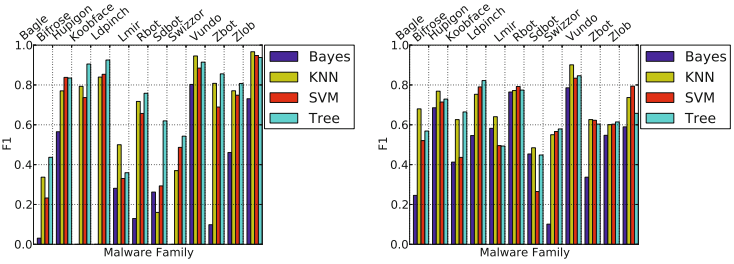
We use the `Orange` [23] software for the four classifiers and the ReliefF algorithm, and `scikit-learn` [32] for the remaining feature selection algorithms. Both Orange and scikit-learn are Python-based general-purpose machine learning software suites. To make our results easily reproducible, we choose to use the original default settings in these software in the experiments. In our tests, we use the five-fold cross validation

method implemented by Orange, which randomly chooses a subset of data containing 80% of the samples for training, and the remaining 20% for testing. We also use the one-against-all method, which builds a classifier for every malware family with samples in this family as positive ones and all samples in all other families as negative ones.

5 Hexdump N-Gram Features

The first type of malware features are collected from outputs of the *hexdump* utility. We use an n-byte sliding window to obtain all possible n-byte sequences inside a binary program and then calculate the frequency of each n-byte sequence. These frequencies are the hexdump n-gram features. Figure 3 depicts the F-1 measures of classification on 1-gram hexdump features by different classifiers.

Imbalanced vs. Balanced. If we consider the imbalanced dataset (see Figure 3(1)), one conclusion may be that malware in the Bagle family are more difficult to classify than those in the Zbot family. In the imbalanced dataset, however, the number of Zbot instances is 8.1 times of that in the Bagle family. If the corresponding balanced dataset is used (see Figure 3(2)), the classification performances are comparable for the two malware families. These observations suggest that *using imbalanced datasets may lead to a distorted conclusion that a malware family is easier to classify than another*. Issues related to imbalanced malware family datasets have also been reported in [29]. In our later experiments, we will only consider balanced datasets.

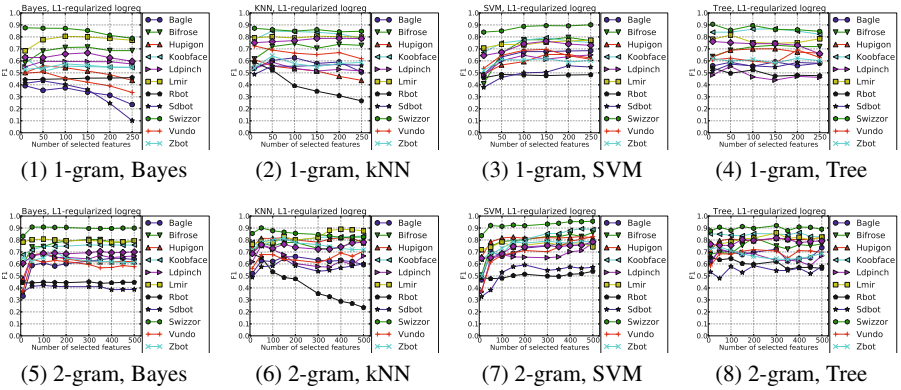


(1) Imbalanced (2) Balanced

Fig. 3. Performances of different classifiers on 256 hexdump 1-gram features

Feature Selection of Hexdump 1-Gram Features. Even though the 256 hexdump 1-gram features are amenable to each of the classification tools, we still want to study whether it is necessary to use all these 256 features for classification. We use the feature selection algorithms discussed in Section 4 to choose the top n features, and use the four classifiers to classify the malware families based on only these top features. We vary n among 10, 50, 100, 150, 200, and 250 in our experiments. The results from the L1-logreg feature selection algorithm are depicted in Figure 4 (1-4).

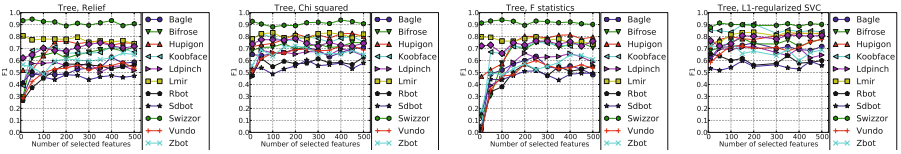
Impact of classification algorithm. We have the following observations. (i) Among all the classifiers, the *decision tree* performs at a high level on a consistent base; also, having more than 10 top features does not improve the classification performance



(1) 1-gram, Bayes (2) 1-gram, kNN (3) 1-gram, SVM (4) 1-gram, Tree

(5) 2-gram, Bayes (6) 2-gram, kNN (7) 2-gram, SVM (8) 2-gram, Tree

Fig. 4. Classification performances with hexdump features (L1-regularized logistic)



(1) Relief (2) χ^2 (3) F statistics (4) L1-SVC

Fig. 5. Feature selection on hexdump 2-gram features (decision tree)

significantly, irrespective of the feature selection algorithm. (ii) When the number of chosen features is small, having more features helps the SVM classifier achieve better classification performance. However, when the number of chosen features grows beyond a certain threshold (e.g., 100), there is little improvement on classification performance with more features. (iii) For most of the malware families, kNN performs at a similar level as the decision tree, but for a few malware families such as Hupigon and Rbot, the performance of kNN deteriorates as the number of selected features grows. (iv) Naive Bayes usually does not perform as well as the other three algorithms. For some malware families such as Swizzor, Bagle and Sdbot, the performance even deteriorates with the number of features selected.

Impact of feature selection algorithm. For most scenarios, the choice of feature selection algorithm does not affect the classification performance much.

Figure 4(4) shows that the top ten features chosen by L1-logreg method are sufficient for the decision tree to reach its peak classification performance on hexdump 1-gram features. The top 10 features are `00`, `40`, `eb`, `24`, `10`, `89`, `8b`, `cc`, `90`, and `ff`.

Feature Selection of Hexdump 2-Gram Features. There are 65,536 hexdump 2-gram features, and it becomes computationally expensive to classify malware based on all them. The feature selection results under different classifiers are shown in Figure 4 (5-8), and Figure 5 depicts the results under different feature selection algorithms.

The key observations are as follows. (i) The decision tree performs consistently well even with a small number ($\leq$ 50) of features chosen by χ^2, L1-logreg and L1-SVC. Under F statistics or Relief, the performance of the decision tree becomes stable only after more than 100 top features, as seen from Figure 5. (ii) The performance of SVM improves as the number of features increases in classification. (iii) When kNN is used, for some malware families, its performance is comparable to the decision tree, but for the Rbot family, its performance even deteriorates as the number of features used for classification increases. (iv) For Naive Bayes, its performance with hexdump 2-gram features becomes more stable than that with hexdump 1-gram ones.

6 Disassembly Code

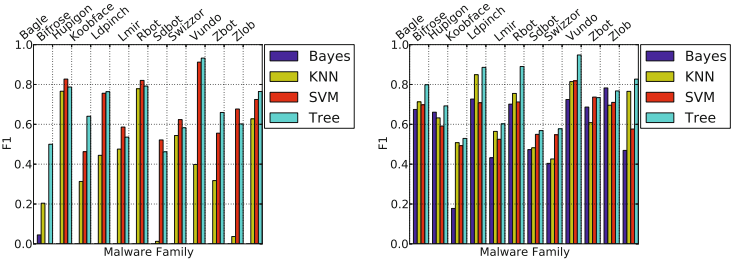
The next type of malware features are extracted from disassembled instructions of malware programs. Disassembly algorithms fall into two categories: *linear sweeping disassembly*, which sequentially resolves instructions that appear in the code section, and *recursive descent disassembly*, which recursively resolves code blocks that start at addresses referenced by other instructions. The standard *objdump* utility uses the linear sweeping algorithm, and for recursive descent disassembly, we implement our own algorithm based on *libdasm* [17]. Note that linear sweeping and recursive descent disassembly apply different disassembly philosophies, and we do not expect that features extracted from one method should be always "better" than the other.

6.1 Objdump

Feature Construction. A typical X86 instruction includes three components: *prefix*, *opcode*, and *operand*. Examples of prefixes include `repne` used for repeating string operations, `cs` which is used for section overriding, `lock` and `wait` which are used to enforce atomic operations. Opcode such as `mov` dictates the action of the instruction, and operand (optional) indicates the data to be operated on. Objdump outputs all three components for each instruction. We concatenate the prefix and opcode components and treat the combination as a feature. For instance, an instruction encoded as `repne scas %es:(%edi),%al` produces a feature as `repne scas`. In total, we create 7259 features, and then calculate their frequencies.

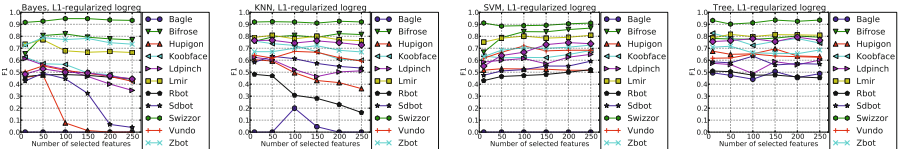
The nature of linear sweeping decides that objdump may dissemble non-code portions. We have seen bad outputs such as `ssssssssssssssssssssssssss`, which is a sequence of `ss`'s. In these cases, we may use features that are semantically meaningless. Even worse, objdump crashes when trying to disassemble some malware executables. Among unpacked instances, objdump can only disassemble successfully 42 Bagle and 135 Ldpinch instances. For the other families, objdump is able to disassemble sufficient samples to populate the balanced dataset.

Classification. The classification performances of using all objdump 1-gram features are shown in Figure 6(1). In most cases, the decision tree and SVM perform similarly well, but for the underrepresented Bagle family, SVM flags every instance as negative. Naive Bayes almost always classifies a positive instance as negative, suggesting that it is inappropriate for malware classification on all objdump 1-gram features.



(1) Objdump (2) Recursive Descent

Fig. 6. Performances of classifiers on disassembly 1-gram features

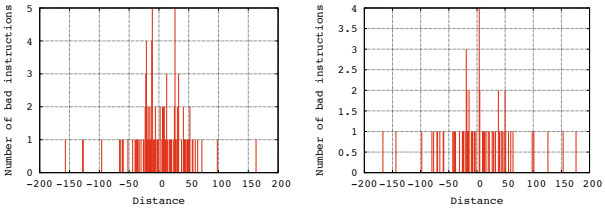


(1) Bayes (2) kNN (3) SVM (4) Tree

Fig. 7. Feature selection on objdump 1-gram features (L1-regularized logistic regression)

Feature Selection. As some of the 7259 features are poor, we select the most discriminative ones from them for classification. Interestingly, there are no significant differences among the results from the five feature selection algorithms. Figure 7 depicts the performances of different classifiers under a varying number of features. Consistent with our observations from Figure 4, performance of kNN deteriorates with the number of features for a few malware families such as Rbot and Hupigon, SVM performs better with more features when the number of features is small, and the decision tree performs well even with a small number of features. Interestingly, Naive Bayes performs better with a small number of features than it does with all the objdump features, suggesting that *including all available features may not pay off for some classification algorithms*.

The top ten features chosen by L1-regularized logistic regression are `lea`, `jmp`, `push`, `add`, `pushl`, `cmp`, `insl`, `mov`, `int3`, and `call`. The `int3` instruction, one of the top features, could be used by malware authors as an anti-debugging technique to thwart reverse engineering efforts. The `insl` instruction, which transfers a string from a port specified in the DX register to the memory address (in 32-bit long) pointed to by the ES:destination index register, could be used by malware for reading network traffic. We notice that these two instructions *do not* appear among the top features if the code is disassembled with the recursive descent algorithm. We plot the number of the closest bad instructions reported by `objdump` against the distance that the closest bad instruction is from every *int3* or *insl* instruction for the malware with md5 239644e31ce940a25a8ca907feba0d19 (a variant of Bagle) and the results are depicted in Figure 8. Many closest bad instructions are indeed close to these two instructions, suggesting that these two instructions are likely generated when `objdump` tries to disassemble non-code portions. This is reasonable because both are single-byte



(1) `int3` (2) `insl`

Fig. 8. Number of closest bad instructions vs. distances from the `int3` and `insl` instructions

instructions, which renders it more likely to be generated when disassembling non-code portions than more complex instructions. This, again, confirms that `objdump` may produce wrong feature values due to its aggressive nature in disassembling.

One may wonder whether there is any correlation among the top hexdump and objdump features. It is noted that the same opcode can be mapped to multiple binary codes. We find that there is *noticeable correlation among the two sets of top features*: seven out of ten top opcode features extracted from objdump outputs can find their corresponding binary codes in the top ten hexdump features.

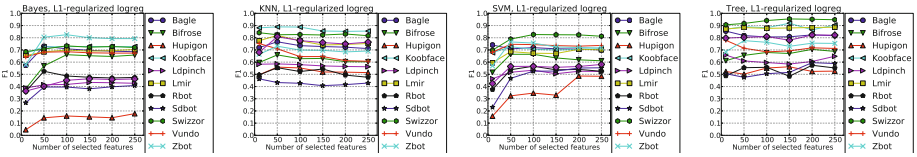
6.2 Recursive Descent Algorithm

From the disassembled instructions of the recursive descent algorithm, we extract only opcodes and count the frequency of each opcode as a feature. In total, we generate 360 1-gram features, much less than those from objdump. We also construct 13,819 2-gram features, each of which is a combination of opcodes in two consecutive instructions.

Classification. Figure 6(2) shows the classification performance based on all 360 recursive descent disassembly features. (i) Clearly, the decision tree outperforms all other three classifiers in almost all the cases. When performing on a much smaller set of recursive descent disassembly features, Naive Bayes does not perform as badly as on objdump features. Actually when classifying the Zbot malware, it outperforms all the other three classifiers. (ii) Our recursive descent algorithm does not crash as objdump often does, and hence we do not have underrepresented malware families. The F-1 measure of the decision tree for classifying Bagle family can be as high as 0.8, which again confirms the importance of class balance in classification. (iii) Although for most families, classification on recursive descent disassembly features performs similarly as on objdump features, there is noticeable performance degradation for the Bifrose family. The recursive descent algorithm is unable to discover code blocks only referenced from indirect jumps, and this may contribute to its worse performance on the Bifrose family, which is known to have adopted encryption and obfuscation techniques to thwart malware analysis [5].

Feature Selection. The impact of feature selection algorithm is little. Figure 9 shows only the classification results using the L1-logreg method. Clearly, for all four classifiers, their performance becomes stable after using only a small number of features.

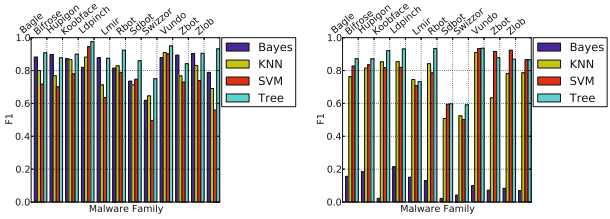
The top ten features chosen by L1-regularized logistic regression are `sub`, `add`, `nop`, `push`, `jmp`, `xor`, `lea`, `call`, `mov`, and `dec`. We observe six of them also



(1) Bayes (2) kNN (3) SVM (4) Tree

Fig. 9. Feature selection on recursive descent 1-gram features (L1-logreg)

appear in the top ten objdump features. Among these top features, the `xor` and `nop` instructions are widely used by malware for obfuscation purpose. We also find there is also noticeable correlation between the top recursive descent and hexdump features.



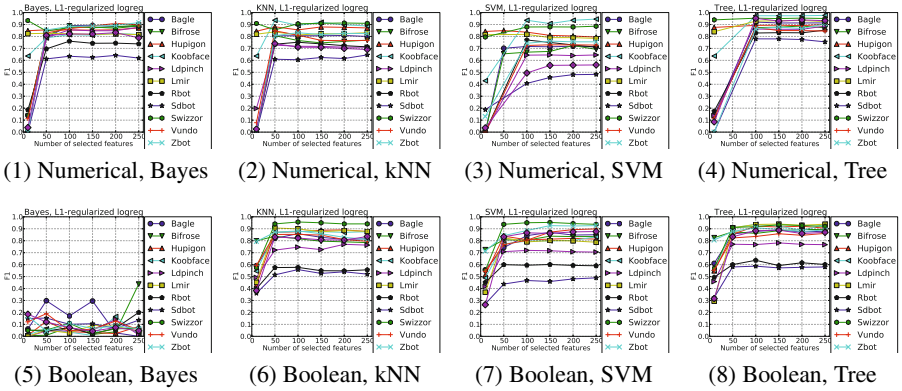
(1) Numerical (2) Boolean

Fig. 10. Performances of different classifiers on PE Header features

7 PE Header

Feature Construction. The PE header is a data structure that describes the meta information of a PE (portable executable) file. It consists of three parts, a 4-byte magic code (always `50 45 00 00`), a 20-byte COFF header containing information such as number of sections and time date stamp, and a 224-byte optional header. The first 96 bytes of the optional header contains information such as major operating system version, size of code, address of entry point, etc, and the remaining 128 bytes are data directories, providing the locations of the export, import, resource, and alternate import-binding directories. We use `pefile` [24] to extract all information from the PE header of an executable program. To construct features from a PE header, we consider two types of information inside it: (1) *Numerical*: almost all fields, except characteristics fields and image resource NameId fields; (2) *Boolean*: every bit of a characteristics field, whether a DLL file is imported or not, and whether a system call in a DLL file is imported or not. In total, we have generated 422 numerical features and 4167 boolean ones.

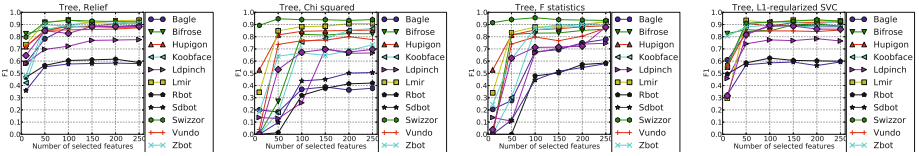
Classification. The classification performances on all numerical or boolean features are shown in Figure 10. We observe that for numerical features, the decision tree almost always performs the best. Interestingly, performance of the Naive Bayes classifier is comparable to the decision tree and for a few malware families (e.g., Bifrose and Vundo), it even performs slightly better. For boolean features, Naive Bayes performs



(1) Numerical, Bayes (2) Numerical, kNN (3) Numerical, SVM (4) Numerical, Tree

(5) Boolean, Bayes (6) Boolean, kNN (7) Boolean, SVM (8) Boolean, Tree

Fig. 11. Feature selection on PE Header features (L1-regularized logistic regression)



(1) Relief (2) χ^2 (3) F statistics (4) L1-regularized SVC

Fig. 12. Comparison of feature selection algorithms on boolean features (decision tree)

the worst among all four classifiers. When working on boolean features, the decision tree performs almost always the best, although in a few cases SVM does slightly better.

Using the decision tree on PE header numerical features, all malware families except Rbot and Sdbot can be classified with high accuracy. This is encouraging for automated malware classification because extracting features from PE headers has a few advantages. First, it does not suffer from those aforementioned challenges associated with disassembling binary code. Second, features from PE headers are easy to extract, and do not require complicated tools such as IDA Pro or a virtual execution environment.

Feature Selection. Figure 11 shows the results of feature selection with the L1-regularized logistic regression method on both numerical and boolean features. It is clear that once the number of features selected goes beyond a certain threshold (e.g., 100), increasing the number of features does not improve the classification performance any more, regardless of the classifier used. This further confirms the importance of feature selection as it is unnecessary to use all features for a classifier to perform well.

We note that with only a small number of numerical features, the decision tree does not produce classification results (see Figure 11 (4)). We have seen the same phenomenon with the other feature selection algorithms. Close examination reveals that this is due to a bug in the C4.5 decision tree implementation.

The impact of the feature selection algorithm is less prominent with numerical features than that with boolean features. Hence, here we only show some results with the latter. Figure 12, together with Figure 11(8), depicts the effects of the feature selection

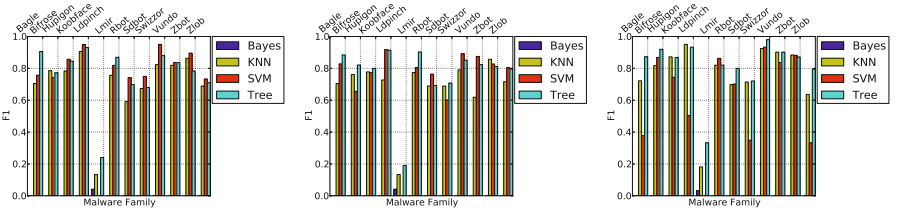
algorithm on classification accuracy. Clearly, the L1-regularized methods perform better than the other algorithms in finding the most discriminative features as they take a smaller number of top features to reach their peak classification performances.

Although PE header features overall possess strong discriminative power, Figures 11 and 12 tell us that with only the top ten features, regardless of whether they are numerical or boolean, none of the classifiers performs well in distinguishing the malware families. We thus do not show the top 10 features here.

8 Dynamic Traces

Feature Construction. We use the Intel Pin [9], a dynamic binary instrumentation tool, to dump a five-minute execution trace for each executable program. However, not all malware programs can finish execution successfully. To create balance among malware families, we use 50 samples for each family in the balanced dataset. Even so, we can only obtain dynamic traces for 46 Lmir, 46 Sdbot and 13 LdPinch samples.

We construct three types of features from it, *opcode 1-gram*, *opcode 2-gram*, and *system calls* from dynamic traces. An opcode 1-gram feature corresponds to the frequency of an opcode (*e.g.*, mov and call) in the trace, and an opcode 2-gram feature to the frequency of a combination of two consecutive opcodes in the trace. A system call feature gives the number of times that a specific system call has been called in the trace.



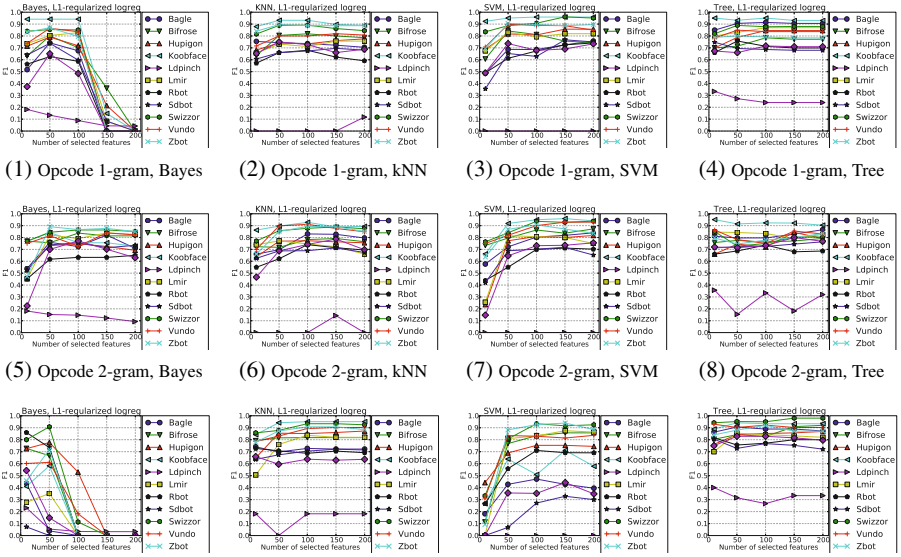
(1) 1-gram (2) 2-gram (3) system call

Fig. 13. Performances of different classifiers on PIN trace features

Classification. Figure 13 depicts the performances of different classifiers on the three types of PIN trace features. It is observed that when working on opcode 1-gram and 2-gram features, classification performances of kNN, SVM, and the decision tree are comparable. But when working on system call features, generally speaking, the decision tree performs better than kNN, which itself outperforms SVM. On the other hand, Naive Bayes performs poorly on all types of PIN trace features, suggesting it is not appropriate for classifying features constructed from PIN traces. It is also noted that classification performance for the LdPinch family is very low due to its underrepresented presence in the evaluation dataset.

Feature Selection. Figure 14 presents the results of feature selection using the L1-logreg method. We note that increasing the number of PIN trace features actually hurts

the classification performance of Naive Bayes. By contrast, SVM performs better with more PIN trace features when the number of features selected is small; when the number of features goes beyond 100, SVM's performance becomes indifferent to the number of features chosen for classification. However, for both the decision tree and kNN, even with as few as 10 features, they still perform well in malware classification.

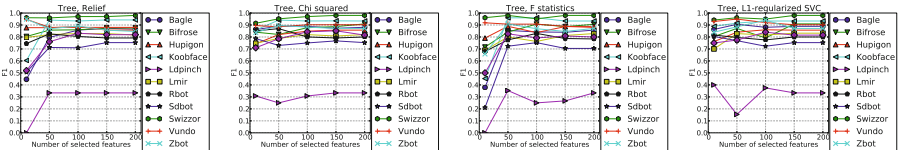


(1) Opcode 1-gram, Bayes (2) Opcode 1-gram, kNN (3) Opcode 1-gram, SVM (4) Opcode 1-gram, Tree

(5) Opcode 2-gram, Bayes (6) Opcode 2-gram, kNN (7) Opcode 2-gram, SVM (8) Opcode 2-gram, Tree

(9) System call, Bayes (10) System call, kNN (11) System call, SVM (12) System call, Tree

Fig. 14. Feature selection on PIN trace features (L1-regularized logistic regression)



(1) Relief (2) χ^2 (3) F statistics (4) L1-Regularized SVC

Fig. 15. Effects of feature selection algorithms with PIN trace system call features

When working on PIN trace opcode n-gram features, the effect of feature selection algorithm is not significant, except that F-statistics does not find the most discriminative features as quickly as the other methods do on PIN trace 2-gram features. Figure 15, together with Figure 14(12), compares the performances of feature selection algorithms on system call features. Clearly, both L1-regularized methods and χ^2 outperform the other two methods in finding the best features for classification quickly.

Table 2. Top 10 PIN trace features

1-gram	add; jmp; mov; cmp; rep movsb; rep movsd; nop; and; xor; push;
2-gram	nop,nop; rep movsb,rep movsb; push,mov; mov,jmp; mov,inc; inc,cmp; jmp,jmp; mov,mov; rep movsd,rep movsd; repne scasb,repne scasb;
System	_strcmpi; RtlInitAnsiString; RtlEnterCriticalSection; KiFastSystemCall; RtlAllocateHeap; RtlFreeHeap; NtSetEvent; RtlInitString; RtlNtStatusToDosError; NtPulseEvent

Table 2 lists the top ten features for each type of PIN trace features. Among the top 10 opcode 1-gram features, we find that around 5-6 features overlap with the top ten objdump or recursive descent features. The two top features that do not appear in the previous lists are `rep movsb` and `rep movsd`, which are used to repetitively move byte and double word from address DS:(E)SI to address ES:(E)DI, respectively. These two instructions could be used by malware to copy large amounts of data in memory. The top system call features include heap related operations (`RtlAllocateHeap` and `RtlFreeHeap`) , mutual exclusion operations (`RtlEnterCriticalSection` and `NtReleaseMutant`), and system calls that allow rootkits to take control of functions calls from user mode to kernel mode (`KiFastSystemCall`).

9 Juxtaposition

Comparison. Figure 16 compares the discriminative power of each type of features in malware classification using the decision tree. The last column of each figure shows the average F-1 measure over all malware families. To make the figures more readable, we show only the results with recursive descent and PIN opcode 2-gram features, as using 2-gram features generally performs as well as, or even better than using 1-gram features. For hexdump, we use only the top 500 2-gram features selected by L1-logreg. For objdump, we use its 1-gram features for comparison. Since for the PIN trace features, the LdPinch family is severely underrepresented, we list in the following table the average F-1 measure when the results for this family are removed when evaluating PIN trace features (for comparison, we also show the average F-1 measures on PE header features including results from the LdPinch family):

type	PIN-2-gram	PIN-SysCall	PE-num (numerical)	PE-bool (boolean)
F-1	0.8110	0.8494	0.8932	0.8426

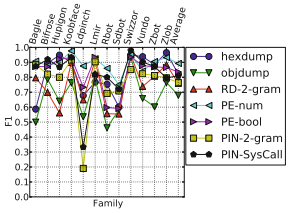


Fig. 16. Comparison of discriminative power

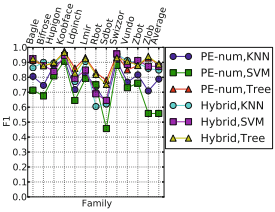


Fig. 17. PE header numerical features vs. hybrid features

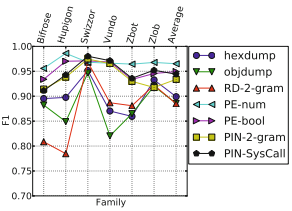


Fig. 18. Comparison results with more samples per family

We have seen that those features extracted from PE headers possess high discriminative power for almost malware families. This is desirable because PE header information is static, which does not require an emulated execution environment, and parsing it does not involve issues related to disassembly. For most malware families, system call features extracted from dynamic traces are also useful for classification. For a few malware families such as Bifrose, Swizzor, and Vundo, using these system call features can classify malware instances slightly better than using those PE header features. System call features extracted from dynamic traces are, however, more difficult to obtain. For instance, we cannot extract PIN trace features from the same number of samples labeled as Ldpinch malware as the other families for fair comparison (see Section 8).

Clearly, some malware families are much easier to detect than the others. For instance, using any type of these features, we are able to classify a Swizzor sample with decent accuracy (F1 measure is at least 0.85). We use IDA Pro to disassemble some Swizzor samples and find that their disassembly code are highly similar, suggesting the Swizzor malware author(s) did not try hard to obfuscate the code. At the other extreme, both Rbot and Sdbot malware are more difficult to detect than the other families. PE header numerical features and PIN trace system call features are the two types of features that are most effective in classifying these two malware families. Even using these features, the F-1 measures can be at most around 0.8. As the source code of Sdbot can be found in the Internet and development of Rbot and other malware has been influenced by it [33], we conjecture that this explains the difficulty of distinguishing Sdbot and Rbot instances observed in our experiments.

PE Header Numerical Features vs. Hybrid Features. Figure 16 tells us PE header numerical features have the most discriminative power in distinguishing malware families. One may wonder whether we can stack up all types of features to improve accuracy of malware classification. Pursuing the answer to this question, however, is again complicated by the class imbalance issue, as for some malware families, we are not able to obtain enough samples with hybrid features from their dynamic traces. To circumvent this challenge, we consider the types of features that ensure that we have 150 samples per family in the balanced dataset. They include: hexdump 1-gram, hexdump 2-gram, PE header numerical, PE header boolean, recursive descent 1-gram, and recursive descent 2-gram. For each of these feature types, we use the top 100 features selected by the L1-regularized logistic regression method.

Figure 17 depicts the classification results based on these features. As a baseline, we also show the classification results using the top 150 PE header numerical features. We do not show the results from Naive Bayes in order to not overcrowd the plots. We observe that for the decision tree classifier, using hybrid features does not affect its performance significantly. From Figure 17, we find that its performance with hybrid features is very close to that when using only the PE header numerical features. For the other two classifiers, they both perform better with hybrid features in most cases. Moreover, since the decision tree is the top performer in most scenarios, none of the malware families can be detected with a much higher accuracy using the hybrid features than using only the PE header numerical features. This suggests that for those malware

families difficult to classify, such as Rbot and Sdbot, it is not sufficient to rely on only those types of features included in the hybrid case for improving their detection rates.

More Samples per Family, Fewer Families. One may wonder whether our down-sampling scheme used to balance different malware families leads to biased conclusions. To verify this, we create a different dataset, which includes samples from only six malware families, `Bifrose`, `Hupigon`, `Swizzor`, `Vundo`, `Zbot`, and `Zlob`. For static analysis features, we use 500 unpacked samples per family, and for dynamic analysis features (i.e., features collected from PIN traces), we use 300 samples per family. It is noted that we exclude the `Rbot` family although it contains 923 unpacked samples (see Table 1). This is because for the `Rbot` family, only 99 unpacked samples successfully generate PIN execution traces. Figure 18 shows the comparison results with the new dataset. Due to fewer families used, the new classification performance is better than that when 12 families are considered. The key observation is that our conclusions drawn previously still hold. For instance, the numerical features extracted from PE headers possess the highest discriminative power. Next to it are boolean features extracted from PE headers and features obtained from PIN execution traces.

10 Discussion

Practical Implications. We hope that our results from this study offer a baseline to compare against for future malware research. When we look for new powerful features to distinguish malware families accurately, we need to check whether they indeed perform better than existing known malware features, particularly when it is a prohibitive process to collect these new features. One interesting observation from this study is that information contained in PE headers possesses high discriminative power in distinguishing the 12 malware families. When we identify a new malware family, although PE header features may not always be indicative of its lineage, we can study them as an early step, given the low cost of obtaining such information. This process can be automated through some feature selection techniques, such as the L1-regularized methods.

Malware programs contain humongous information we can leverage for automated malware classification. One may want to build a computationally powerful classifier that is able to process all available information in hope of optimizing classification accuracy. Such an approach may not work well, as even for the same type of features, including all features may not boost the classification performance for some classifiers. Due to this fact of more-is-not-always-better, it is important to evaluate the sensitivity of classifiers to the number of features used in automated malware classification.

Rethinking Ground Truth Data. One biggest hurdle for malware research is how to obtain ground truth data. This study relies on the ground truth data we can obtain through consensus among AV software. Our observations made from this work, such as the high discriminative power of PE header information and system calls invoked in dynamic execution, hold true when we build a classifier that distinguishes the 12 malware families in the midst of some benign programs. In practice, we will encounter samples that do not belong to these 12 families. For the purpose of cross validation, we examine another well-known labeled malware dataset, which has been used in [4]. This dataset contains 2,658 malware variants, among which 2,332 are detected to be unpacked by

PEiD. The authors use a reference clustering technique to cluster them into 84 malware families. We, however, notice that the sizes of the clusters in this labeled malware dataset are highly skewed and except the top three clusters, the others contain only a few samples. Hence, this dataset is inappropriate for validating our conclusions here. We welcome the community to use other labeled datasets to verify our observations.

Revisiting Methodology. This study reveals that L1-regularized methods, a type of embedded methods, perform consistently well in feature selection. The is because filter methods rank individual features only independently, and hence, a group of features that are highly ranked *individually* may not achieve good discrimination performance *collectively* due to their correlation. By contrast, L1-regularized methods aim to find a subset of features that minimize the loss functions collectively.

Our study has tested only four widely used classifiers, and shown that a variant of decision tree performs well in automated malware classification. There may be other classifiers that perform better than these four. For instance, we can apply ensemble of classifiers (e.g., AdaBoost) to further improve the classification accuracy. Even for the four classifiers considered, we can further tune the parameters to achieve better performance. Even though conducting an exhaustive comparison of different classifiers is out of the scope of this work, our observation that the decision tree classifier can achieve good classification performance on a consistent basis, as well as the fact that decision tree has scalability advantages over other classifiers, suggests that practical deployment of automated malware classification should take it into serious consideration.

It is noted that as discriminatory as features extracted from PE headers are, they cannot fully replace other types of features, particularly those from dynamic analysis. For instance, more complicated dynamic malware behavior analysis, such as that done by Anubis [2], could produce powerful features for automated malware classification. However, the classification methodology adopted in this study, which assumes features represented as vectors of numerical or boolean values, may not fully reveal the discriminative power of features extracted from Anubis analysis. For instance, string-level information produced by Anubis, such as locations and names of files created, read, or written by a malware instance, which could be useful for malware classification, cannot be easily incorporated into our analysis framework. Due to these concerns, we leave detailed analysis of features from Anubis analysis as our future work.

It is an arms race between malware authors and cyber defenders. The theme of this study is to study the discriminative power of malware features, and even though some features of malware are highly indicative of their lineages, it is possible that malware authors manipulate these features to confuse automated malware classification. Robustness of features is an important issue [37], and we can imagine that for some feature types, such as PE header information, could be more easily manipulated than others such as the system calls invoked in dynamic execution. An automated malware classifier can combine multiple feature types extracted from malware programs to improve its robustness. Moreover, building an automated malware classification system should be a dynamic process, and if we witness new malware samples in which some features are manipulated to confuse classification, we should update the automated malware classification system by incorporating these new samples into the training dataset.

Acknowledgment. We acknowledge discussions with Daniel Quist, Marian Anghel, and Tanmoy Bhattacharya, and are grateful to Christopher Kruegel and Paolo M. Comparetti for the labeled dataset used in [4].

References

1. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T.: Graph-based malware detection using dynamic analysis. Journal of Computer Virology 7(4), 247–258 (2011)
2. http://anubis.iseclab.org/
3. Bailey, M., Oberheide, J., Andersen, J., Mao, Z.M., Jahanian, F., Nazario, J.: Automated classification and analysis of internet malware. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 178–197. Springer, Heidelberg (2007)
4. Bayer, U., Comparetti, P.M., Hlauschek, C., Kruegel, C., Kirda, E.: Scalable, behavior-based malware clustering. In: NDSS 2009 (2009)
5. http://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj Bifrose-ZI/detailed-analysis.aspx
6. Canali, D., Lanzi, A., Balzarotti, D., Christoderescu, M., Kruegel, C., Kirda, E.: A quantitative study of accuracy in system call-based malware detection. In: ISSTA (2012)
7. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering 21 (2009)
8. Hu, X., Chiueh, T.-C., Shin, K.G.: Large-scale malware indexing using function-call graphs. In: CCS 2009 (2009)
9. http://www.pintool.org/
10. Jang, J., Brumley, D., Venkataraman, S.: Bitshred: feature hashing malware for scalable triage and semantic analysis. In: Proceedings of ACM CCS 2011 (2011)
11. Kolbitsch, C., Comparetti, P.M., Kruegel, C., Kirda, E., Zhou, X., Wang, X.: Effective and efficient malware detection at the end host. In: USENIX Security 2009 (2009)
12. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. Journal of Maching Learning Research 7, 2721–2744 (2006)
13. Kong, D., Ding, C., Huang, H., Zhao, H.: Multi-label relieff and f-statistic feature selections for image annotation. In: IEEE CVPR 2012 (2012)
14. Kononenko, I.: Estimating attributes: analysis and extensions of relief. In: Bergadano, F., De Raedt, L. (eds.) ECML 1994. LNCS, vol. 784, pp. 171–182. Springer, Heidelberg (1994)
15. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 207–226. Springer, Heidelberg (2006)
16. Li, Y.: Building a Decision Cluster Classification Model by a Clustering Algorithm to Classify Large High Dimensional Data with Multiple Classes. PhD thesis, The Hong Kong Polytechnic University (2010)
17. http://code.google.com/p/libdasm/
18. Liu, H., Li, J., Wong, L.: A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns. Genome Informatics 13 (2002)
19. Maggi, F., Bellini, A., Salvaneschi, G., Zanero, S.: Finding non-trivial malware naming inconsistencies. In: Jajodia, S., Mazumdar, C. (eds.) ICISS 2011. LNCS, vol. 7093, pp. 144–159. Springer, Heidelberg (2011)
20. Microsoft security intelligence report (January-June 2006)
21. Nataraj, L., Yegneswaran, V., Porras, P., Zhang, J.: A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: ACM AISec 2011 (2011)
22. http://www.offensivecomputing.net/ (accessed in March 2012)

23. http://orange.biolab.si/
24. http://code.google.com/p/pefile/
25. Perdisci, R., Lanzi, A., Lee, W.: Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In: ACSAC 2008 (2008)
26. Raman, K.: Selecting features to classify malware. In: Proc. of InfoSec Southwest (2012)
27. Rieck, K., Krueger, T., Dewald, A.: Cujo: efficient detection and prevention of drive-by-download attacks. In: ACSAC 2010 (2010)
28. Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic analysis of malware behavior using machine learning. J. Comput. Secur. 19(4), 639–668 (2011)
29. Rossow, C., Dietrich, C.J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., van Steen, M.: Prudent practices for designing malware experiments: Status quo and outlook. In: IEEE Symposium on Security and Privacy (May 2012)
30. Roth, V., Lange, T.: Feature selection in clustering problems. In: NIPS 2004. MIT Press, Cambridge (2004)
31. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Proc. of IEEE Symposium on Security and Privacy (2001)
32. http://scikit-learn.org/
33. http://www.honeynet.org/node/53
34. Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: PE-Miner: Mining structural information to detect malicious executables in realtime. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 121–141. Springer, Heidelberg (2009)
35. http://www.symantec.com/about/news/release/article.jsp?prid=20110404_03
36. https://www.virustotal.com/
37. Yang, C., Harkreader, R.C., Gu, G.: Die free or live hard? Empirical evaluation and new design for fighting evolving twitter spammers. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 318–337. Springer, Heidelberg (2011)
38. Ye, Y., Wang, D., Li, T., Ye, D., Jiang, Q.: An intelligent pe-malware detection system based on association mining. Journal in Computer Virology (2008)
39. Yu, H.-F., Huang, F.-L., Lin, C.-J.: Dual coordinate descent methods for logistic regression and maximum entropy models. Machine Learning 85(1-2), 41–75 (2011)