Hierarchical Learning for Automated Malware Classification

Shayok Chakraborty[⋆], Jack W. Stokes[†], Lin Xiao[†], Dengyong Zhou[†], Mady Marinescu[‡] and Anil Thomas[‡]
[⋆]Arizona State University, Tempe, AZ 85281 USA
[†]Microsoft Research, One Microsoft Way, Redmond, WA 98052 USA
[‡]Microsoft Corp., One Microsoft Way, Redmond, WA 98052 USA

***Abstract*—Despite widespread use of commercial anti-virus products, the number of malicious files detected on home and corporate computers continues to increase at a significant rate. Recently, anti-virus companies have started investing in machine learning solutions to augment signatures manually designed by analysts. A malicious file's determination is often represented as a hierarchical structure consisting of a type (e.g. *Worm, Backdoor*), a platform (e.g. *Win32, Win64*), a family (e.g. *Rbot, Rugrat*) and a family variant (e.g. *A,B*). While there has been substantial research in automated malware classification, the aforementioned hierarchical structure, which can provide additional information to the classification models, has been ignored. In this paper, we propose the novel idea and study the performance of employing hierarchical learning algorithms for automated classification of malicious files. To the best of our knowledge, this is the first research effort which incorporates the hierarchical structure of the malware label in its automated classification and in the security domain, in general. It is important to note that our method does not require any additional effort by analysts because they typically assign these hierarchical labels today. Our empirical results on a real world, industrial-scale malware dataset of 3.6 million files demonstrate that incorporation of the label hierarchy achieves a significant reduction of 33.1% in the binary error rate as compared to a non-hierarchical classifier which is traditionally used in such problems.**

***Index Terms*—Hierarchical Machine Learning, Automated Malware Classification**

I. Introduction

While the advent of the Internet has revolutionized communication, business and the access of information, it has also allowed attackers to craft and control malware that targets vulnerabilities in remote computer systems. As a result, malware detection has been an active research area over the past two decades [1]. In many cases, malware signatures designed by analysts and deployed in commercial anti-virus products continue to provide the majority of detections on users' computers. However, signature generation is an expensive process in terms of time and human labor, mainly due to the staggering number of malware samples that are automatically generated by attackers on a regular basis. Our company receives hundreds of thousands of unknown files each day that are not detected by antivirus signatures. This figure indicates that there is a pressing need for automated techniques to reliably analyze and interpret unknown software to ensure the safety of computing systems. To address this fundamental need, anti-virus companies have been investing in machine learning classifiers [2], [3], [4] which learn to distinguish between malicious and legitimate files for future unseen instances based on features extracted from static and dynamic analysis. A malware label typically consists of a hierarchical structure with a type (*Trojan*, *PWS*), a platform (*Win32*, *Win64*), a family (*Rbot*, *Rugrat*) and, optionally, a variant (*A, B*). Examples of two malware names are depicted in Figure 1. The behavioral nature of a malicious file becomes more specific as we traverse the label hierarchy. The malware type specifies the coarse distinguishing nature of the file. For example, *Backdoors* provide a hidden communication mechanism for remote attackers, and *Trojans* typically include a small amount of malicious code inserted into a legitimate program. Attackers often target specific platforms such as Win32 or Win64 based on prevalence and the underlying security mechanisms built into the operating system. New malware families names are assigned to files which belong to the overall broad malware type, but are considered distinct for some reason. For example, bots such as *Rbot*, *Zbot*, and *Sdbot* are typically assigned to the *Backdoor* type but differ enough to warrant their own family name. Family differences within a type may be due to similar, but unique, malicious code written by independent malware authors. Finally, the variant represents a small variation within a family. To avoid detection, malware authors sometimes modify their code several times per day depending on the frequency of new malware signatures released by the anti-virus companies. In other cases, malware authors may share the underlying source code which is then modified to some degree by the new group. When existing signatures fail to detect the new family variation, analysts may create a new variant name. A variant may be created to correspond to a specific malware "generic" signature which attempts to detect many difference instances of a specific type of polymorphic malware using a small set of features.

The key question we address in this paper is: Can we utilize the extra hierarchical label information that analysts are *already providing* to improve malware classification? A number of common machine learning algorithms, including classification [5], [6], [7], [8], [9], [10], clustering [11], [12], [13] and association rule mining [14], have been proposed to distinguish malicious samples from legitimate ones. However, none of these techniques exploit the inherent hierarchical nature of the malware labels. The hierarchical label name provides useful information to the underlying classification models, which can potentially improve the classification accuracy. Malware analysts typically provide the label hierarchy

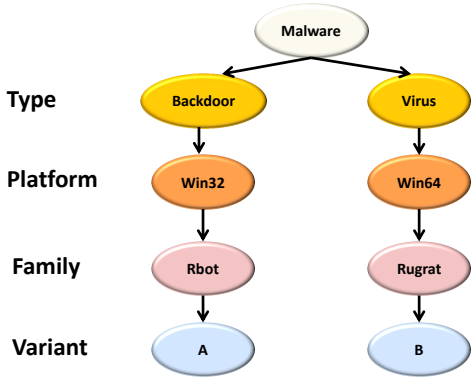


Fig. 1. Hierarchical structure of two example malware labels consisting of a type, platform, family and variant.

during file annotation; exploiting the hierarchical structure can thus augment valuable information to the classification models without any additional manual effort.

In this paper, we propose a novel system which uses hierarchical learning for automated malware classification. Hierarchical classification algorithms have been shown to perform well in the context of information retrieval, text mining and other applications [15], [16], [17], [18]. We study the performance of a hierarchical support vector machine (SVM)-based learning algorithm, Orthogonal Transfer [18] for the automated classification of malicious portable executable (PE) files. We selected this hierarchical learning algorithms based on its superior results in the machine learning literature [18]. To the best of our knowledge, this is the first attempt to exploit the hierarchical nature of malware files to improve the classification accuracy and to use hierarchical learning in the security domain, in general. Following the standard practice of hierarchical learning testing [15], [18], we also compare and contrast the performance of the hierarchical learning algorithms against a flat (i.e. non-hierarchical) SVM. Flat models are conventionally adopted in malware classification research. We demonstrate on an extremely large dataset of 3.6 million files that inclusion of the label hierarchy *significantly* reduces the binary error rate over traditional flat classification. Our specific contributions are as follows: (1) We incorporate the hierarchical structure of the malware labels, consisting of a type, platform, family and variant, in its automated analysis. We use state-of-the-art hierarchical learning frameworks for this purpose; (2) We validate the performance of the algorithms with 2.5 million training samples and 1.1 million test samples. This is the largest scale empirical validation of these hierarchical algorithms and (3) Our empirical results demonstrate a significant reduction in the binary error rate, as compared to flat SVM, by incorporating the label hierarchy in the learning framework.

II. Label Tree Construction

We construct the label tree using a top-down strategy. For each sample, the type, platform, family, and variant are derived from the hierarchical malware labels, and the training data includes examples belonging to the 33 malware types. Starting with the types, we next append nodes representing the platforms and so on. We found that some malware families or variants were extremely rare and had very few instances. Training a multi-class classifier with very few samples in a class may lead to poor performance. We therefore decided on a threshold taken as 10,000 in our experiments, and all malware leaf nodes not meeting this criteria were put in a *Default* node under the same malware parent node. An unknown file that does not have a malware type, platform, family or variant represented in the training set can also be assigned to a *Default* node during prediction.

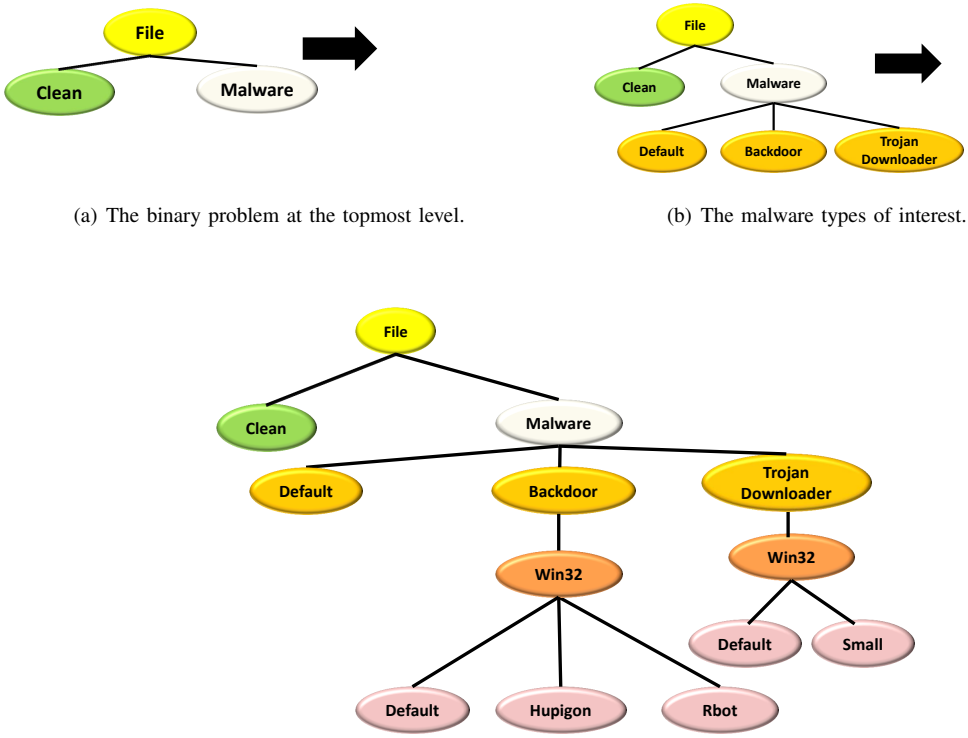
After node thresholding, 52 malware families were retained. The typical tree depth of the resulting label tree for malware without (with) variants is 5 (6) including the root. The tree depth for benign files is 2. Therefore when possible, we use the maximum depth in the labels according to the leaf threshold. Further, the distribution across malware types is not balanced. Our objective in this work is to demonstrate the performance of hierarchical learning on a real-world malware dataset. The data we obtained from our company's anti-malware products reflects this real-world distribution which is unbalanced in practice.

This method of label tree construction further allows us to study the effects of specific malware types; for instance, one may want to focus only on the malware types *TrojanDownloader*, *Backdoor* and *Adware*. In this case, the families, and in some cases the variants, of these malware types form the leaves of the tree and any other malware type is put in a *Default* malware node at that level. An example of the label tree construction is depicted in Figure 2.

III. Dataset and Feature Representation

The raw data obtained for this study was created by Microsoft's anti-malware product team by modifying the production engine used in the Windows operating system. As part of the scanning process, an unknown file is first emulated in the anti-malware engine to determine its behavior before being run natively using the underlying operating system. In addition to the hierarchical label, we were provided two types of raw data collected during file emulation, including null-terminated objects identified in the emulated process memory and system API calls and their input parameters, for each file. Typically, the null-terminated objects are strings which have been unpacked as the malware executes in the emulator, but sometimes they correspond to sections of executable code which happen to contain a null-terminated pattern. In addition, the emulator extracts the sequence of system API calls and their associated parameters as the file is being emulated. An example of a portion of this event stream for a Windows portable execution (PE) file is shown in Table I.

The two sets of raw logs (Unpacked File Strings and System API Calls Plus Parameters) allows us to create three types of features for this study, namely, unpacked files strings, trigrams of system API calls, and distinct combinations of a system API call and an ordered parameter value. For the first set of features utilized in our system, the *Unpacked File Strings* are used directly. From the event stream, the *API Call*



(a) The binary problem at the topmost level.

(b) The malware types of interest.

(c) A portion of the complete label tree.

Fig. 2. Method of constructing the top-down label tree.

System API Call Event Stream
KERNEL32.DLL!GetVersion()
KERNEL32.DLL!HeapCreate(0, 4096, 0)
NTDLL.DLL!RtlAllocateHeap(0x00480000, 0, 320, 0x00000140)
KERNEL32.DLL!InitializeCriticalSection(0x0046b060)
KERNEL32.DLL!InitializeCriticalSection(0x0046b090)

TABLE I
AN EXAMPLE OF THE SYSTEM API CALL EVENT STREAM.

Trigram features are determined as unique combinations of three consecutive system API calls. For the example system API call stream, the first potential API trigram is (GetVersion, HeapCreate, RtlAllocateHeap) and the second is (HeapCreate, RtlAllocateHeap, InitializeCriticalSection). The *System API plus Parameter Value* features are determined by unique combinations of a system API call and a parameter such as HeapCreate with the second parameter, dwInitialSize, set to a value of 4096 bytes in Table I.

The product team provided us the raw logs from approximately 3.6 million files. We next constructed the feature representation for these examples which we then randomly split into two separate datasets including 2.5 million for training and 1.1 million for test. The original data contains 70.1% malware and 30.9% benign files. Extracting all feature types from the training files generated over 50 million potential features. We employ feature selection, using mutual information [19], to reduce the potential features to approximately 265 thousand sparse binary features in order to avoid overfitting. That is, each file is represented as a row in a sparse binary array, and the appropriate column is set to one if the file contains the corresponding feature. A sparse data format is used for the data array in order to efficiently represent the file in memory. The original training set includes over 13,800 distinct malware family labels, however, not all files are assigned to an explicit class representing their family or variant because most families or variants did not meet the required 10,000 file threshold described in the previous section.

IV. ALGORITHMS

SVMs have shown good results in several studies on malware classification literature [5], [10]. Furthermore, hierarchical learning has mostly been studied in the context of support vector machines [15], [16], [17], [18]. To answer the primary question, *"Does using the existing label hierarchy improve malware classification?"*, in a controlled study, we fixed the base classification algorithm as the SVM and the optimization method (RDA). We therefore use the SVM to isolate the improvement of hierarchical learning for malware classification.

We study the performance of the *Orthogonal Transfer* hierarchical learning algorithm proposed by Zhou *et.al.* [18]. This method is selected because of its superior empirical performance, as reported in [18]. Since this method uses the SVM as the base classifier, we use the flat multi-class SVM

learning algorithm proposed by Crammer and Singer [20] as the baseline comparison. Our main goal is to precisely quantify how the label hierarchy affects the malware classification accuracy. For fair comparisons, the original optimization problem in both approaches is first transformed into an unconstrained problem by eliminating the slack variables, and solved using the regularized dual averaging (RDA) method [21] [22]. Our methodology for comparing these algorithms matches that from the machine learning community [18]. By using the same RDA optimization technique and underlying hinge loss in the SVM, any performance improvements are due to the hierarchical algorithm and labeling structure. We next describe the algorithms at a high-level, where we adopt the mathematical notation from [18]. The interested reader is encouraged to consult the original paper which proposed each algorithm.

Baseline Algorithm: Flat Multi-Class SVM: The flat multi-class SVM proposed by Crammer and Singer [20] learns a mapping from the feature vectors $\{(\mathbf{x}_1), \cdots, (\mathbf{x}_N)\}$ of the N training examples to their labels $\{y_1, \cdots, y_N\}$ where $y_k \in \mathcal{Y}$ and $\mathcal{Y}$ is the set of possible labels. For the *Flat SVM*, the labels in $\mathcal{Y}$ only correspond to the leaf nodes in the label tree in Figure 2(c); the label tree structure is ignored. In some cases, the label may be a malware variant. In other cases, the label may be a specific *Default* class in the label tree. Each legitimate file is assigned the "Clean" label. The main objective behind the SVM is to maximize the margin which is the distance from each class' decision boundary to its support vectors. Only samples which are mispredicted contribute to the weight vector updates. Correctly predicted samples do not affect the training in any way. The *Flat SVM* learns a separate hyperplane, with weight vector $\mathbf{w}_i$, for each class, and the predicted class label for an unknown file is selected as the one which has the largest score. The optimization problem for the *Flat SVM* is:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\sum_{i\in\mathcal{Y}} \|\mathbf{w}_i\|^2 + \frac{C}{N}\sum_{k=1}^{N}\xi_k \\ \text{subject to} \quad & \mathbf{w}_{y_k}^T\mathbf{x}_k - \mathbf{w}_i^T\mathbf{x}_k \geq 1-\xi_k, \\ & \qquad \forall\, i \in \mathcal{Y}\setminus\{y_k\},\ \forall\, k \in \{1,\ldots,N\}, \\ & \xi_k \geq 0,\ \forall\, k \in \{1,\ldots,N\}. \end{aligned} \tag{1}$$

The *Flat SVM* has two terms in the objective function. The right-hand term computes the hinge loss associated with each sample k and minimizes the number of misclassifications. The slack variables ξ are greater than zero if the sample is mispredicted (i.e. the predicted class label does not match the true label) and zero otherwise. The left-hand term provides regularization and favors solutions where the sum of the L2 norms of the weight vectors associated with each leaf node is small. C is a trade-off parameter controlling the relative importance of the two terms. The constraint imposes the condition that the classifiers $\mathbf{w}$ should be trained in such a way that for each training sample $\mathbf{x}_k$, the prediction score output by the model $\mathbf{w}_k$ corresponding to its actual label y_k is higher (at least by a threshold $1-\xi_k$) than the prediction scores given by all the other models.

Hierarchical Learning Algorithm: Orthogonal Transfer: This algorithm was proposed by Zhou *et al.* [18], and each node in the hierarchical label tree has a classifier $\mathbf{w}_i$. The rationale of this algorithm is to enforce the classifier at each node to be maximally different from the classifiers of its ancestors; this way, the classifier at each node can capture different information compared to those of its ancestors. To achieve this goal, the classifier hyperplane at each node is orthogonal (i.e. diverse) to all of its ancestors. Specifically, the mathematical formulation is as follows:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\sum_{i,j=1}^{m} K_{ij}\left|\mathbf{w}_i^T\mathbf{w}_j\right| + \frac{C}{N}\sum_{k=1}^{N}\xi_k \\ \text{subject to} \quad & \mathbf{w}_i^T\mathbf{x}_k - \mathbf{w}_j^T\mathbf{x}_k \geq 1-\xi_k,\ \forall\, j \in \mathcal{S}(i), \\ & \qquad \forall\, i \in \mathcal{A}^+(y_k),\ \forall\, k \in \{1,\ldots,N\}, \\ & \xi_k \geq 0,\ \forall\, k \in \{1,\ldots,N\}. \end{aligned} \tag{2}$$

The objective function is similar to (1) but has some key differences. In the context of this hierarchical classification, the labels in $\mathcal{Y}$ are identified as all nodes in the category tree. In addition to minimizing the Euclidean norm of the weight vector, as in the *Flat SVM*, the left-hand regularization term ensures the orthogonality condition by the term $|\mathbf{w}_i^T\mathbf{w}_j|$. The symmetric matrix $K \in \mathcal{R}^{m\times m}$ (with $K_{ij} \geq 0$ for all $i, j = 1\ldots m$) encodes the hierarchical structure embedded in the tree, and K_{ij} is set to 0 whenever node i is neither an ancestor nor a dependent of node j. So, the orthogonality condition holds for every node only for its set of ancestors. The slack variable based term has a similar interpretation as in the *Flat SVM* algorithm. The constraints also have a similar interpretation as the *Flat SVM* but only consider a nodes' siblings, $\mathcal{S}(i)$. As in [18], $\mathcal{A}(i)$ are the ancestors (i.e. parents) of node i not including itself, while $\mathcal{A}^+(i)$ does include the node i. Neither $\mathcal{A}(i)$ nor $\mathcal{A}^+(i)$ include the root node. We used the Regularized Dual Averaging (RDA) algorithm with optimality bound from [18] to solve the optimization problem. Please refer to [18] for more details about the solution technique. To classify a test instance, we proceed level by level, where at each level, the linear classifier generating the maximal score among its siblings is selected, and that path is used to proceed lower down the tree, until a leaf node is reached.

V. Experimental Results

In this section, we empirically compare the performance of the hierarchical learning algorithm against the *Flat SVM* baseline algorithm. Each learning algorithm is trained on the labeled training data, and its performance is evaluated on the unseen test set. The parameter C in the SVM formulation is taken as 100 in all our experiments. Our preliminary investigations affirmed the fact that it is not possible to train a model on the *entire* 2.5 million samples. We therefore used the efficient *mini-batch* training approach in our experiments to reduce the training time. In this technique, the entire training data is split randomly into many batches of smaller size, and the algorithms are trained incrementally on each batch. When all the batches are covered in this way, a single pass or an *epoch* is said to have been completed on the training data. For

subsequent epochs, the data is randomly shuffled again and new mini-batches are selected. The mini-batch size is taken as 10,000, and the training process repeated for 100 epochs where the solution from one epoch is used as the starting point for the next epoch. The best solution is selected based on the accuracy on a held-out validation set. Each mini-batch is randomly split into 70% for the actual mini-batch training set and 30% for the validation set.

A. Binary Classification Results

The most important aspect of automated malware classification is differentiating malware from legitimate files. The malware vs. benign detection results are depicted in Table II

Algorithm	Binary Error Rate (%)
Flat SVM	3.90
Orthogonal	2.61

TABLE II
BINARY ERROR RATES IN PERCENTAGES. THE ERROR RATE FOR ORTHOGONAL TRANSFER IS MUCH LOWER THAN THE FLAT SVM BASELINE.

We see that the *Orthogonal Transfer* algorithm yields lower error rate (2.61%) compared to the *Flat SVM* (3.90%). Thus, by utilizing the label hierarchy, *Orthogonal Transfer* is able to significantly reduce the binary error rate by 33.1% compared to the *Flat SVM* without any additional manual effort.

We also evaluate the binary performance of the learners in terms of the precision, recall, and F-Score in Table III. We note that *Orthogonal Transfer* offers the highest precision (95.51%) and F-Score (95.78%). *Orthogonal Transfer*'s recall (96.06%) is only slightly less than that of the *Flat SVM* (97.96%).

Algorithm	Precision (%)	Recall (%)	F-Score (%)
Flat SVM	89.32	97.96	93.44
Orthogonal	95.51	96.06	95.78

TABLE III
BINARY MALWARE DETECTION RESULTS. ORTHOGONAL TRANSFER HAS THE HIGHEST PRECISION AND F-SCORE. ITS RECALL IS SLIGHTLY LESS THAN FLAT SVM.

As reported in Table IV, we next compute the false positive rate (FPR) and the false negative rate (FNR) for the binary labels. In this case, a file is determined to be a false positive if its true label is benign but is predicted by the classifier to belong to any malware family or variant. Anti-virus companies are much more concerned about false positives than false negatives since removing a legitimate file may yield the computer inoperable. The FPR for the *Orthogonal Transfer* framework is much lower than the *Flat SVM* baseline, which is critical.

B. Computation Time Analysis

Besides measuring the performance on the test sets, we also report the time required to train the algorithms on 2.5 million samples. The training times (in hours) to complete 100 epochs for the two algorithms studied in this paper are as follows: **Flat SVM:** 165.75, **Orthogonal:** 61.31. These results indicate that the *Orthogonal Transfer* algorithm is computationally much more efficient in terms of training time on a per epoch basis than the *Flat SVM* algorithm. It is also important to note that while training requires days, all software is implemented in C# in a single thread. The training time can be decreased by implementing the system in C++ with multi-threading and further reduced with acceleration by a GPU. Furthermore, evaluating an unknown file using the *Orthogonal Transfer* classifier is very fast due to computing the sum of several *sparse*, linear classifiers at each level.

Algorithm	FPR (%)	FNR (%)
Flat SVM	4.73	2.04
Orthogonal	2.02	3.94

TABLE IV
FALSE POSITIVE RATES (FPRS) AND FALSE NEGATIVE RATES (FNRS) IN PERCENTAGES. THE FPR FOR ORTHOGONAL TRANSFER IS MUCH LOWER THAN THE FLAT SVM BASELINE.

VI. RELATED WORK

Related work falls into two main categories including malware classification and hierarchical machine learning. We provide a summary of some of the important related work in both areas.

Malware Classification: Given the significant threat to users, malware classification has been an active area of research recently. Ideka and Mathur provide a good overview of previous work in this field in [1]. Kephart [5] proposed a neural network-based system for automated malware classification. Schultz *et al*. [9] compared the malware detection performance of naive Bayes and multi-naive Bayes with signatures and an inductive rule-learner. Kolter and Maloof [6] evaluated naive Bayes, decision trees, support vector machines, and boosting approaches for malware classification. Mody and Lee [13] described a k-means approach for malware family clustering. Bayer *et al.* [11] exploited locality sensitive hashing (LSH) techniques to efficiently assemble sets of malware samples based on the nature of their threats. Rieck et al. [10] studied malware classification of malware families using a support vector machine. Their approach is similar to our baseline, but differs in several ways. They analyzed a corpus of 10,072 samples compared to 3.6M in our study. In addition, [10] does not analyze the binary classification results.

Recently, El-Bakry [23] used an efficient Time Delay Neural Network (TDNN) implemented in the frequency domain using Fast Fourier Transforms (FFTs) for sequential malware classification. Ye *et al.* [24] proposed a hierarchical associative classifier for detecting malware in a gray list. Although this system includes the term hierarchical in the name, this framework merely uses a two-stage, rule-based classifier to improve detection performance; it does not utilize the label hierarchy of the malware samples or hierarchical learning. Jang *et al.* [12] used

the Jaccard coefficient as the similarity metric together with feature hashing for large-scale, high dimensional automated malware analysis. A sequential approach using Hidden Markov Models (HMMs) for malware classification was presented by Muhaya *et al.* [25]. Karampatziakis [3] improved the baseline classification of individual files included in containers (e.g. .zip and .rar files) using two logistic regression models and a bi-partite graph of containers and files. Based on a value of information clustering criterion and confidence-weighted linear classifier, Neugschwandtner *et al.* [8] improved sample selection for automated analysis. Kong and Yan [7] presented a malware family classifier which learned a metric distance to discriminatively separate files belonging to these families.

Hierarchical Machine Learning: Hierarchical classification has mostly been studied in the context of the support vector machine (SVM) classifier. A good overview of the standard SVM formulation can be found in [26], [27]. Dumais and Chen [17] proposed using the SVM for hierarchical learning and evaluated a two layer hierarchy on a large web corpus. Dekel *et al.* presented a large margin hierarchical classifier [16]. Bianchi *et al.* [28] introduced the H-loss function for training hierarchical classifiers based on the principle that an erroneous prediction at a given node in the hierarchy should not result in additional penalty in the subtree of the node. The same authors also proposed the B-SVM algorithm for hierarchical classification which approximated the Bayes optimal classifier with respect to the H-loss and demonstrated improved performance over hierarchical SVMs [29]. Cai and Hofman [15] proposed the Tree Loss algorithm for hierarchical learning. Zhou [18] demonstrated good hierarchical classification performance using their Orthogonal Transfer algorithm, which is evaluated in this work in the context of large-scale malware classification. Due to its wide usage in hierarchical classification, we adopt the SVM as the base classifier in this work. The specific variant of SVM used in this paper was published by Crammer and Singer [20].

VII. Conclusions

In this paper, we explore the usage of hierarchical learning algorithms for large-scale malware classification. The malware label hierarchy is provided by the analysts during file annotation. To the best of our knowledge, this is the first research effort to exploit the label hierarchy to improve malware classification accuracy without any additional extra human effort. We tested the performances of one hierarchical learning algorithm and one flat classification strategy on a real world malware data with 2.5 million training samples and 1.1 million test examples. As mentioned previously, we use the type, platform, family and variant-based label information of a malware sample in this work. The *Orthogonal Transfer* algorithm significantly outperforms the *Flat SVM* in terms of the binary class error rate. This hierarchical algorithm also offers significant improvement over flat classification in terms of the false positive rate, precision, and F-Score. As part of our future work, we plan to exploit the usage of multiple cores and cluster nodes to decrease the training time.

Acknowledgment: The authors thank Dennis Batchelder for his support and guidance during this project.

References

[1] N. Idika and A. Mathur, "A survey of malware detection techniques," Purdue Univ., Tech. Rep., February 2007.
[2] D. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining and inference for malware detection," in *SDM*, 2011.
[3] N. Karampatziakis, J. Stokes, A. Thomas, and M. Marinescu, "Using file relationships in malware classification," in *DIMVA*, 2012, pp. 1–20.
[4] Symantec, "Machine learning for anti-virus software," http://www.aboutdm.com/2013/04/machine-learning-for-anti-virus-software.html.
[5] J. O. Kephart, "A biologically inspired immune system for computers," in *International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life)*. MIT Press, 1994, pp. 130–139.
[6] J. Kolter and M. Maloof, "Learning to detect and classify malicious executables in the wild," *JMLR*, pp. 2721–2744, 2006.
[7] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *KDD*, 2013, pp. 1357–1365.
[8] M. Neugschwandtner, P. M. Comparetti, G. Jacob, and C. Kruegel, "Forecast: Skimming off the malware cream," in *Annual Computer Security Applications Conference (ACSAC)*. IEEE Computer Society Press, 2011.
[9] M. Schultz, E. Eskin, and E. Zadok, "Data mining methods for detection of new malicious executables," in *IEEE Symposium on Security and Privacy*, 2001.
[10] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov, "Learning and classification of malware behavior," in *DIMVA*, 2008.
[11] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2009.
[12] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: Feature hashing malware for scalable triage and semantic analysis," in *CCS*, 2011.
[13] T. Lee and J. Mody, "Behavioral classification," in *Annual Conf. of the European Institute for Computer Antivirus Research (EICAR)*, 2006.
[14] W. Li, J. Han, and J. Pei, "Cmar: Accurate and efficient classification based on multiple class-association rules," in *IEEE ICDM*, 2001.
[15] L. Cai and T. Hofmann, "Hierarchical document categorization with support vector machines," in *CIKM*, 2004.
[16] O. Dekel, J. Keshet, and Y. Singer, "Large margin hierarchical classification," in *ICML*, 2004.
[17] S. Dumais and H. Chen, "Hierarchical classification of web content," in *SIGIR*, 2000.
[18] D. Zhou, L. Xiao, and M. Wu, "Hierarchical classification via orthogonal transfer," in *ICML*, 2011.
[19] C. D. Manning, P. Raghavan, and H. Schutze, *An Introduction to Information Retrieval*. Cambridge University Press, 2009.
[20] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *JMLR*, vol. 2, pp. 265–292, 2001.
[21] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Mathematical Programming*, vol. 120, pp. 221–259, 2009.
[22] L. Xiao, "Dual averaging methods for regularized stochastic learning and online optimization," *JMLR*, vol. 11, pp. 2543–2596, 2010.
[23] H. El-Bakry, "Fast virus detection by using high speed time delay neural networks," *Journal in Computer Virology*, vol. 6, pp. 115–122, 2010.
[24] L. Ye, T. Li, K. Huanga, Q. Jiang, and Y. Chen, "Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list," *Journal of Intelligent Information System*, vol. 35, pp. 1–20, 2010.
[25] F. Muhaya, M. Khan, and Y. Xiang, "Polymorphic malware detection using hierarchical hidden markov model," in *IEEE DASC*, 2011.
[26] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines*. Cambridge University Press, 2000.
[27] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.
[28] N. Bianchi, C. Gentile, A. Tironi, and L. Zaniboni, "Hierarchical classification: Combining bayes with svm," in *NIPS*, 2005.
[29] N. Bianchi, C. Gentile, and L. Zaniboni, "Hierarchical classification: Combining bayes with svm," in *ICML*, 2006.