Unsuccessful Story about Few Shot Malware Family Classification and Siamese Network to the Rescue

Yude Bai[1], Zhenchang Xing[2], Xiaohong Li[1*]
Zhiyong Feng[3], Duoyuan Ma[1]
[1]Tianjin Key Laboratory of Advanced Networking (TANK), School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China
[2]Research School of Computer Science, Australian National University, Data61 CSIRO, Australia
[3]School of Computer Software, College of Intelligence and Computing, Tianjin University, Tianjin, China
{baiyude,xiaohongli,zyfeng,2018216033}@tju.edu.cn,zhenchang.xing@anu.edu.au

ABSTRACT

To battle the ever-increasing Android malware, malware family classification, which classifies malware with common features into a malware family, has been proposed as an effective malware analysis method. Several machine-learning based approaches have been proposed for the task of malware family classification. Our study shows that malware families suffer from several data imbalance, with many families with only a small number of malware applications (referred to as few shot malware families in this work). Unfortunately, this issue has been overlooked in existing approaches. Although existing approaches achieve high classification performance at the overall level and for large malware families, our experiments show that they suffer from poor performance and generalizability for few shot malware families, and traditionally downsampling method cannot solve the problem. To address the challenge in few shot malware family classification, we propose a novel siamese-network based learning method, which allows us to train an effective MultiLayer Perceptron (MLP) network for embedding malware applications into a real-valued, continuous vector space by contrasting the malware applications from the same or different families. In the embedding space, the performance of malware family classification can be significantly improved for all scales of malware families, especially for few shot malware families, which also leads to the significant performance improvement at the overall level.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Malware family classification, Few shot learning, Siamese network

*Xiaohong Li is the corresponding author.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380354

ACM Reference Format:
Yude Bai, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, Duoyuan Ma. 2020. Unsuccessful Story about Few Shot Malware Family Classification and Siamese Network to the Rescue. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380354

1 INTRODUCTION

Android system is the most common target by malware due to its popularity [McAfee-Labs 2018]. A recent threat report shows that about 5 millions of new mobile malware is captured from Jan-2018 to Jan-2019, which is an increase of 31% over the previous year [Nokia 2019]. As the manual analysis of new malware is time demanding [Zhang et al. 2013], the malware analysis is increasingly overwhelmed by the number of newly disclosed malware. Due to the polymorphic property of Android malware, many newly disclosed malware is just polymorphic variants of existing malware [Zhou and Jiang 2012], [Feng et al. 2014]. As a result, grouping malware with common behavior into malware families and studying the signature of the malware families have been proposed as an effective means of battling the fast growth of new mobile malware [Fan et al. 2018b].

Security analysts manually create several malware family datasets, such as Android Malware Genome Project (Genome) [Zhou and Jiang 2012] (referred to as M-1 in this work) that has 1245 malware applications in 33 families, Drebin Dataset (Drebin) [Arp et al. 2014] (M-2) that has 5416 malware applications in 131 families, and AMD Project (AMD) [Wei et al. 2017] (M-3) that has 24478 malware applications in 71 families. The AMD dataset has a variant that splits some top-level families into sub-families, and this variant (M-3s) has 24478 malware applications in 134 families. A multi-class malware-family classifier can be trained using features of malware applications in these datasets to predict the family label of a new malware application. A wide range of syntactic features have been exploited in this classification task, such as Android permissions [Wang et al. 2014], critical API calls [Deshotels et al. 2014], and attributes of inter-component communication (ICC) [Avdiienko et al. 2015].

We represent a malware application as a multi-dimensional one-hot feature vector, each dimension of which corresponds to the presence or absence of one syntactic feature in this application. Based on the literature survey of effective malware-identifying features [Au et al. 2012; Feng et al. 2016; Wang et al. 2014], we select 250 syntactic features, including 50 Android permissions,

156 critical API calls and 44 ICCs. We then build malware-family classifiers on the datasets M-1, M-2, M-3 and M-3s using the normal model training method (see Section 4.1), by which the classifier is trained using certain percentage of the labeled data in a dataset and then tested on the rest data. As shown in Figure 3, a simple k-nearest neighbor (KNN) search [Roussopoulos et al. 1995] with "max-win" strategy can achieve the F1-score 0.951 on M-1, 0.825 on M-2, 0.854 on M-3 and 0.806 on M-3s. Training the Support Vector Machine (SVM) [Hsu and Lin 2002], Decision Tree (DTree) [Safavian and Landgrebe 1991] or Random Forest (RF) [Breiman 2001] classifiers can achieve comparable or even better overall performance on these four datasets.

However, under this glory overall performance, one critical data characteristic of the malware family datasets has been overlooked. That is, the malware family datasets have severe data imbalance across malware families. As shown in Figure 2, except a small number of large malware families, there is a long tail of medium and small malware families. Take the M-3 dataset as an example. The largest family has 7842 malware applications, while the smallest family has only 4 malware applications. The median family size is 24, the first quartile is 12 and the third quartile is 171. In this work, we consider a malware family with more than 100, 10 to 100 (inclusive), and less than 10 malware applications as a large, medium and small malware family, respectively. By this family scale definition, the total numbers of malware applications in large, medium and small families have comparable distributions in the four malware family datasets, so that we can investigate the performance on the three different scales of families across the datasets.

When we look into the prediction performance for different scales of malware families, the unsatisfactory performance, especially for small families, emerges from the shadow of the glory overall performance. Take the M-3 as an example, the F1-score by the SVM for the large, medium and small families are 0.916, 0.876, and 0.600 respectively. As the small malware families have the small numbers of malware applications, their classification errors do not significantly impact the overall performance. However, such classification errors do matter from malware analysis perspective, as they could lead to ineffective prevention efforts [Fan et al. 2018c; Tang et al. 2019]. Another critical issue shadowed by the glory overall performance is the high variance of the model performance across the 10-fold cross validations (see Figure 5). For the SVM on M-3, 21 medium and small families have the F1-score variance larger than 0.3. In these 21 families, the SVM have the lowest F1-score 0 and the highest F1-score 1 in different runs. These large performance variance indicates that the classifiers obtained by the normal model training method is sensitive to the choice of training data and suffer from poor generalizability.

The above model performance and generalization issues reveal the fundamental limitation of the current malware family classifier training methods, especially for the few shot learning in the medium and small families where only a small number of malware applications are available for model training. Unfortunately, the severe data imbalance across malware families renders downsampling and upsampling methods ineffective in improving the prediction performance on medium and small malware families or improving the generalization of the classification model [Chawla et al. 2004; He and Garcia 2008]. Our experiments also confirm this (see Figure 3 and Figure 5).

In this paper, we propose a siamese-network [Bromley et al. 1994; Zagoruyko and Komodakis 2015] based learning method to address the performance and generalization issues caused by the severe data imbalance in malware family datasets. Intuitively, a siamese network transforms the multi-class classification of a malware application into the binary classification of whether two malware applications come from the same family. A siamese network consists of two identical, weight sharing MultiLayer Perceptron (MLP) (i.e., feature extractors) [Bertinetto et al. 2016; Koch et al. 2015] which should learn to embed malware applications into a continuous vector space, in which malware applications in the same family should be close but those from different families should be apart. Training the siamese network requires pairs of malware applications: those from the same family are positive training data, while those from different families are negative training data. A large number of positive and negative training pairs can be generated, even for small malware families. This leads to effective training of the feature extractor in the few shot learning setting. The MLP-based feature extractor trained by the siamese network can then be connected to a multi-class classifier to predict the family label of unseen malware applications.

We conduct extensive experiments to compare our siamese-network based learning method and the normal model training method on the four malware family datasets. Our experiment results show that the classifiers obtained by siamese-network based learning perform statistically significantly better than those obtained by normal model training method, even though siamese-network based learning uses only a small percentage (e.g., 11.20% for M-3) of all data used in normal model training method. Furthermore, siamese-network based learning can significantly boost the performance of malware family classification for all scales of families. The smaller the family is, the larger it can boost the performance. As such, it narrows the performance gap between the large and small families. Combining downsampling with siamese-network based learning can improve the model generalizability, especially for small families. Finally, our comparative studies confirm that these benefits cannot be obtained through simple downsampling method or MLP-based feature extractor, without using the siamese-network based learning method.

Our work makes the following new contributions:

- To the best of our knowledge, our work is the first to systematically investigate the severe data imbalance issue in malware families and the impact of this issues on the performance and generalizability of malware family classification, especially in the few shot learning setting.
- We conduct extensive experiments and analysis to show that neither downsampling method nor MLP-based feature extractor in the normal model training setting can successfully address the performance and generalizability issues caused by the severe data imbalance issue in malware families.
- We innovatively adopt the siamese-network based learning for the task of malware family classification, and conduct comparative experiments which confirm the superior performance of our siamese-network based learning over normal

model training method, not only at the overall level but also at all families scales, especially for few shot learning.

2 RELATED WORK

Android malware classification has two broad types: binary (benign/malware) classification [Allix et al. 2016; Avdiienko et al. 2015; Pendlebury et al. 2019; Wang et al. 2014] and multi-class malware family classification [Deshotels et al. 2014; Feng et al. 2014; Garcia et al. 2018]. Our work focuses on the second type.

Both types of the classification tasks require an effective feature representation of the applications to be classified. Many simple syntactic features and complex semantic features (i.e. graph-based features) have been identified in existing studies [Avdiienko et al. 2015; Deshotels et al. 2014; Feng et al. 2016; Wang et al. 2014]. Wang et al. [Wang et al. 2014] performs benign/malware classification based on Android permission. MudFlow [Avdiienko et al. 2015] detect malware behaviors by syntactic features (API, Intent) via data flow analysis by FlowDroid [Arzt et al. 2014]. DroidLegacy [Deshotels et al. 2014] classify piggybacked malware applications into families by API call graph for malware applications. Astroid [Feng et al. 2016] constructs a maximally suspicious common subgraph of inter-component call graph for each malware family, and then adopts an approximate graph matching algorithm to determine the family of a malware application. Although semantic features can be more predictive than syntactic features, they are more expensive to compute and hard to define. In contrast, syntactic features can be easily extracted from Android applications by the tools like AudroGudard [Desnos et al. 2011], and thus are widely used for malware family classification [Avdiienko et al. 2015; Feng et al. 2016; Wang et al. 2014].

In this work, we also adopt syntactic features, including 50 permissions, 156 APIs and 44 attributes of inter-component communication, to represent malware applications. A naive feature representation is one-hot vector. Each vector element represents the presence or absence of a syntactic feature in the applications. In addition, we also use real-valued, neural feature vectors computed by the MLP-based network [Collobert and Bengio 2004] from the naive one-hot feature vectors. Many studies [Gao et al. 2019; Kim 2014; McLaughlin et al. 2017], including those on malware and vulnerability analysis [Guo et al. 2019; Han et al. 2017; Yuan et al. 2014], show that neural feature extractors can outperform naive one-hot feature vector, because neural feature extractors can better encode the important features and their implicit relationships. However, our study shows that neural feature extractors computed by a not-well-trained MLP network may even lead to the performance degradation in malware family classification. Our proposed siamese-network based learning is an effective means of training the MLP-based feature extractor.

The adoption of siamese network is driven by the severe data imbalance across malware families. In such imbalanced classification tasks, classes with few instances include less information than other large classes [Wang and Yao 2019]. Siamese network is one of the effective few shot learning methods, aiming to acquire knowledge from few data instances through contrastive pairs. It was introduced in early 1990s for a few shot image matching problem [Bromley et al. 1994]. Since then, it has been adopted in many few shot computer vision tasks [Bertinetto et al. 2016; Snell et al. 2017; Wang and Hebert 2016; Zagoruyko and Komodakis 2015]. To the best of our knowledge, our work is the first to use siamese network for the malware family classification task, and we zoom into different scales of malware families and investigate the effectiveness of siamese-network based learning for few shot malware family classification. In contrast, existing machine learning methods [Fan et al. 2018b; Suarez-Tangil et al. 2014; Zhang et al. 2014] for the malware family classification task adopt normal model training method and investigate the classification performance only at the overall level.

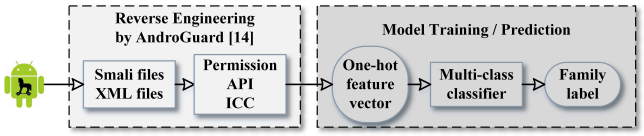

Figure 1: The task of malware family classification.

A remotely related line of research to our work is fault or defect prediction. Malhotra et al. [Malhotra 2016] uses object-oriented metrics for defect prediction on Android operating system by 18 machine learning methods, which confirms the predictive capability of machine learning methods. HIRER [Fan et al. 2018a], based on the high-frequency keywords extracted from Abstract Syntax Trees, is proposed to learn the functional and semantic information from Android application source code for predicting defective files. However, our work is different from fault prediction in two fundamental ways. First, our goal is to predict the family label of a malware application, while fault prediction is to predict which parts of the code is error prone. Second, fault prediction uses normal machine learning method and it does not have to deal with few shot learning issue. In contrast, we show that the malware family classification suffers from few shot learning issue, and our work explicitly addresses this issue by siamese-network based learning.

3 MALWARE FAMILY PREDICTION

As shown in Figure 1, we formulate the malware family prediction task as a multi-class classification problem, that is, given a malware application, classify it into one of three or more malware families. The malware applications are represented as a multi-dimensional feature vector reverse-engineered from the applications. A machine learning classifier is trained using a dataset of malware applications labeled with their corresponding malware families. The prediction performance of the classifier is tested with malware applications unseen during model training, and is evaluated in terms of accuracy, precision, recall and F1-score.

3.1 Syntactic Features of Malware Applications

In this work, we adopt syntactic features[1] for malware analysis, including 50 Android permissions, 156 critical API calls and 44 attributes of ICC, due to their effectiveness in malware family prediction [Deshotels et al. 2014; Feng et al. 2014; Wang et al. 2014] and the easiness of decompiling these syntactic features from mobile applications [Chen et al. 2018, 2016]. We extract these syntactic

[1]Please visit https://github.com/qWe1aSd/malPre for the list of these features in details.

features using the AndroGuard tool [Desnos et al. 2011] from the Smali and XML files of an application.

3.1.1 Android Permission. Android permission gives fine-grained security features to restrict specific operations of an application [Wang et al. 2014]. For example, an Android application needs *android.permission.READ_SMS* to read SMS messages, and it needs *android.permission.INTERNET* to open network sockets. Android provides an attribute (i.e., pretectionLevel) for each permission, which can be "normal", "dangerous", "signature", and "signatureOrSystem". The "normal" permissions have lower risk, while the "dangerous" permissions have higher risk. After excluding the system-granted permissions with the "signature" and "signatureOrSystem" attributes [Wang et al. 2014], we collect 50 Android permissions used by the malware applications in our datasets.

3.1.2 API Call. Sensitive data can be accessed by specific API calls, such as *getAccount* to get access to user accounts and *updateNetwork* to update system network time. By tracing and analyzing the call of these APIs, it is possible to characterize the benign or malicious behaviors of an application. We therefore leverage the PScout [Au et al. 2012], which produces a mapping between permissions and API calls, to identify the APIs related to the selected 50 permissions. Finally, 156 critical APIs are identified for malware family prediction.

3.1.3 Inter-component Communication (ICC). An Android application consists of four components: Activity, Service, BroadcastReceiver, and ContentProvider. Intent, as an abstract description of an operation to be performed, is employed to accomplish the inter-component communication. In our work, we consider types of Intent attributes, which contains 44 ICC features [Avdiienko et al. 2015]. First, *Action* defines the general action to be performed, for example, *android.intent.action.DIAL* to dial a phone number. Second, *Category* provides additional information about the action to executed, for example, *android.intent.category.HOME* as the first activity displayed. At last, few critical Intents are selected from [Avdiienko et al. 2015] such as *android.service.wallpaper.WallpaperService* to set live wallpaper.

3.2 Multi-Class Malware-Family Classifiers

We use four types of multi-class classifiers that are commonly used for malware family prediction in the literature [Breiman 2001; Hsu and Lin 2002; Roussopoulos et al. 1995; Safavian and Landgrebe 1991], including k-Nearest Neighbor (KNN), Support Vector Machine (SVM), Decision Tree (DTree), and Random Forest (RF). A classifier takes as input the feature vector of a malware application and produces as output the family label of this application. The input vector can be either the one-hot feature vector of the application's syntactic features or the neural feature vector computed by the neural feature extractor (see Section 5.1) from the one-hot syntactic feature vector. In this work, we investigate two types of classifier training methods: normal model training method (see Section 4.1) versus siamese-network based learning method (see Section 5.1), and study the impact of the two methods on the overall performance and the performance of different scales of malware families.

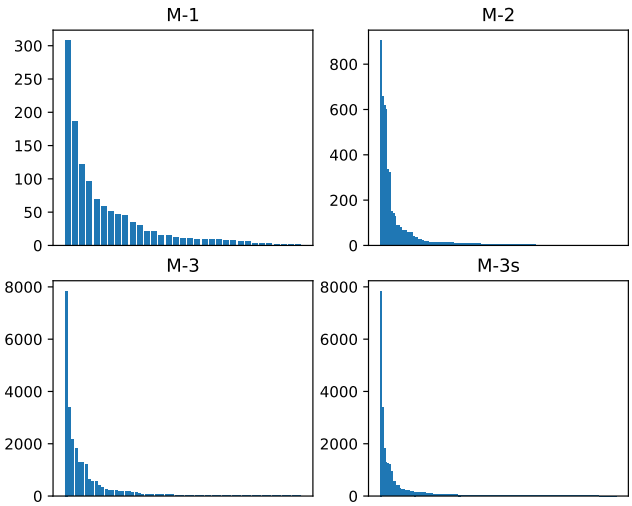

Figure 2: Distribution of the size of malware families (X-axis: malware families[2]; Y-axis: family size).

KNN is a non-parametric model for classification. It retrieves the k-nearest training malware applications of a new malware application by the Euclidean distance in the feature space, and then determines the family label of the new malware application based on the family labels of these k-nearest neighbors by a weighted max-win strategy. A SVM is a discriminative classifier formally defined by a separating hyperplane. Given labeled malware applications (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new malware applications. A DTree is a tree-like model of the features of malware applications and their family labels, where leaves represent family labels and branches represent conjunctions of features that lead to those family labels. RF is an ensemble learning method for classification, by constructing multiple decision trees for different random subsets of features. The family label of a new malware application is produced by the voting result ("Max Wins" strategy) of all decision trees.

3.3 Datasets

We carry out our experiments on three well-known datasets of Android malware families: Android Malware Genome Project (Genome) [Zhou and Jiang 2012], Drebin Dataset (Drebin) [Arp et al. 2014] and AMD Project (AMD) [Wei et al. 2017]. Genome collects more than 1200 malware applications that cover 49 Android malware families. Drebin contains 5560 Android applications from 179 different malware families [Arp et al. 2014]. AMD contains 24553 malware applications, which is categorized in 71 malware families [Wei et al. 2017]. AMD also has a variant that splits some families into sub-families, and this variant has 136 families.

Besides the applications that can not be decompiled successfully, we also remove the families with only one application of malware applications from these datasets, because they cannot be used for model training and testing. Then we obtain four datasets M-1, M-2 M-3 and M-3s respectively for our experiments. M-1 consists of 1245

[2]Due to space limitation, we omit the family names. Distribution charts with family names are available at https://github.com/qWe1aSd/malPre.

malware applications in 33 malware families. M-2 contains 5416 malware applications in 131 malware families. M-3 has 24478 malware applications in 71 malware families. M-3s has 24478 malware applications in 134 malware families. Figure 2 shows the family size (i.e., the number of malware applications) of malware families in these four dataset. We can find that the family size of all the four datasets meet the long-tail power-law distribution, which reveals the severe imbalance of the family sizes in malware family datasets. For example, the largest family *Airpush* in M-3 has the 7842 malware applications, while the smallest family *Fobus* in M-3 has only 4 malware applications. We refer to these datasets as the "all" dataset in the performance comparison.

Downsampling is a common method to address the data imbalance issue [He and Garcia 2008]. Note that oversampling is not adopted because we can not judge whether a newly created malware represent a meaningful malware and which malware family it should belong to. To study the impact of downsampling on the model performance (overall and per family scale), we create a downsampling dataset for M-1, M-2, M-3 and M-3s datasets, respectively. We refer to these downsampling dataset as the "part" dataset in the performance comparison. We cannot simply downsample all families to the size of the smallest families, because it will discard most of the data. Instead, we follow the stratified downsampling strategy for different family scales [ElRafey and Wojtusiak 2017; Sun et al. 2009]. Specifically, we downsample the large families to 100 malware applications for each large family, and the medium families to 10 malware applications for each medium family. Small families remain intact. The resulting downsampling datasets have 42.89%, 30.50%, 11.20% and 14.55% of the malware applications in M-1, M-2, M-3 and M-3s datasets, respectively. They have 48.62%, 23.23%, 9.84% and 12.05% of the malware applications in the large families of the four datasets, respectively, and have 28.88%, 36.29%, 35.96% and 33.88% of the malware applications in the medium families of the four datasets, respectively. The small families have 13.86%, 18.28%, 2.63%, and 6.82% of the malware applications in the four downsampling datasets, which is much more balanced than 5.94%, 5.58%, 0.29%, and 0.99% in the original datasets.

We do not downsample the large and medium by a fixed percentage of the family size, because this will make some large families into medium families, and make some medium families into small families, depending on the downsampling percentage. In that way, although we can still compare the overall performance between the original dataset and the corresponding downsampling ones, we cannot compare the performance by the family scale between the original dataset and the corresponding downsampling ones, because a downsampling dataset has different sets of large/medium/small families from those of the same family scales in the original dataset.

3.4 Evaluation Metrics

We adopt Accuracy, Precision, Recall and F1-score which are widely used to evaluate the classification models [Stehman 1997], [Xu et al. 2016], [Xia et al. 2016], [Wang et al. 2016]. **Accuracy** (**ACC**) refers to the correct classification results out of all the malware applications in the testing dataset. **Precision** (**P**) for each family f_i is defined as the proportion of malware applications which is correctly classified into f_i divided by the number of malware applications classified as f_i. **Recall** (**R**) for each family f_i is defined as the proportion of malware applications which is correctly classified into f_i divided by the number of malware applications labeled as f_i. **F1-score** (**F1**) is the harmonic mean of Recall and Precision for each family. All experiment results in this work are obtained by 10-fold cross validation (see Section 4.1). For each run, the evaluation metric is the weighted average of that metric over all malware families (or one scale of families - large, medium, or small) in the testing data. We then average the evaluation metrics over the 10 runs as the performance on a dataset. Due to space limitation, the performance comparison uses F1-score in the paper. But all evaluation metrics are available at https://github.com/qWe1aSd/malPre.

4 UNSUCCESSFUL STORY OF FEW SHORT MALWARE FAMILY PREDICTION

In this section, we examine the overall performance of the malware family classifiers obtained through the normal model training method, look into the performance differences for large, medium and small malware families, and study the impact of downsampling on the overall performance and the performance of different scales of malware families. Our results confirm the unsuccessful story of few shot malware-family prediction by the normal model training method, even though the models' overall performance is acceptable.

4.1 Normal Model Training Method

We apply 10-fold cross validation in all our experiments. The malware applications of a large or medium family are randomly split into 10 equal-size subsets. The remainders left (if any) are randomly put into these subsets (one per subset). For the small families with N ($N < 10$) malware applications, we first adopt the leave-one-out strategy to obtain N-fold cross validation data. Then, we randomly duplicate $10 - N$ folds of these N folds. The original N folds and the duplicate $10 - N$ folds constitute the 10-fold cross validation data for the small families. In this way, we guarantee at least 1 sample for each family in the testing data.

We use 9 subsets of data as the training data, and one subset as the testing data. We denote this normal model training method as "nosia", as opposed to siamese-network based learning method (see Section 5.1). We first split the downsampling dataset into 10-fold training/testing data for the experiments on the "part" datasets. Then, we split the data left in the original dataset after downsampling into 10 subsets. These 10 subsets contain only the malware applications in the large and medium families. We add these subsets of large and medium family data to the corresponding "part" training data for training the model in "all" setting. In this way, the testing data is the same in the "all" versus "part" settings for across-setting comparison.

4.2 Model Hyperparameters Optimization

We adopt the grid search method [Ndiaye et al. 2018] with a 5-fold cross validation for optimizing the hyperparameters of the four classifiers. We use M-1 for this optimization purpose. As M-1 is the smallest dataset, we have low risk of model overfitting for the other three larger datasets. After model optimization, the same classifier settings are used for the experiments on all the four datasets.

Table 1: Performance Metrics on M-1

		ACC	P	R	F1
nosia_part	SVM	0.927	0.889	0.927	0.904
	DTree	0.9	0.862	0.9	0.875
	RF	0.948	0.92	0.948	0.932
	KNN	0.94	0.92	0.94	0.926
nosia_all	SVM	0.943	0.914	0.943	0.926
	DTree	0.903	0.866	0.903	0.879
	RF	0.95	0.925	0.95	0.934
	KNN	**0.96**	**0.946**	**0.96**	**0.951**

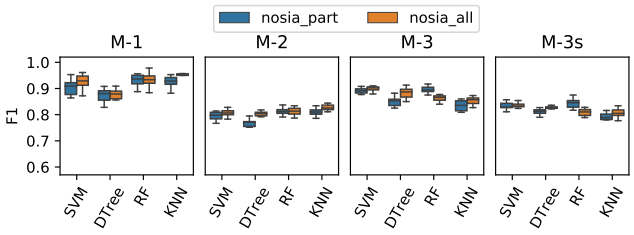

Figure 3: Overall performance: nosia_all versus nosia_part. All sub-figures share the same y-axis.

We use the SVM with RBF (Radial Basis Function) kernel function [Vert et al. 2004]. We experiment the penalty parameter C and the kernel parameter γ_{svm} with the values $\{0.001, 0.01, 0.1, 1, 3, 5, 10\}$ as search space, and finally set the two parameters at 10 and 0.01, respectively. For DTree, we set *max_depth* (the maximum depth of decision tree) at 157 after experimenting the *max_depth* from 1 to 250, and we apply information gain as well as "best" split strategy at each node. In RF, we set *max_depth* (the maximum depth of decision tree) at 20 and *n_estimators* (the number of decision trees in forest) at 150, after experimenting the two parameters with the values $\{5, 10, 20, 40, 80, 100, 150, 200, 250\}$. For the KNN, we set k (the number of neighbors contributing to the classification decision) at 2 after experimenting the k from 1 to 10.

4.3 Results

***RQ1*: What is the overall performance of the four malware-family classifiers obtained through normal machine learning methods? What is the impact of downsampling on the overall performance?**

Table 1[3] summarizes the performance metrics of the four classifiers obtained by the normal model training method ("nosia") on the original ("all") M-1 dataset versus the downsampling ("part") M-1 dataset. We can see that a simple KNN classifier in the nosia_all setting achieves the best overall performance, with the accuracy (ACC) 0.960, precision (P) 0.946, recall (R) 0.960 and F1-score (F1) 0.951 on average. Our experiment results are consistent with existing studies that also use the M-1 dataset for malware family prediction, Dendroid [Suarez-Tangil et al. 2014], Astroid [Feng et al. 2016], and DroidSIFT [Zhang et al. 2014], which report the prediction accuracy 0.942, 0.938 and 0.930, respectively. Note that these existing studies do not report precision, recall and F1-score. Furthermore, they do not conduct experiments on the other three datasets we use. But the SVM, DTree and RF classifiers we use replicate their classification methods.

Figure 3 visualizes the overall performance in both the nosia_all and nosia_part settings. Box-plots show the weighted average F1-score of a classifier over all families in a dataset in the 10 runs of cross validation. Therefore, we have 16 classifier-dataset combinations for each setting. The same figure format is used to visualize the performance results in other RQs.

Figure 3 shows that KNN also performs very well on M-2 (a medium-size dataset). But SVM outperforms the other three classifiers on the largest dataset M-3 and M-3s. Across the four datasets, we do not see an always-winner when comparing the two classifiers. In fact, the average F1-scores of the four classifiers on a dataset are close with the minor differences about 0.072 ± 0.022. Comparing the performance across the four datasets, we find that the number of malware families in a dataset (i.e., the number of classes to predict) affects the performance the most. When the number of malware families in the datasets are close (e.g., M-2 versus M-3s), the less data a dataset has (M-2 in this comparison), the poorer the prediction performance is.

Comparing the performance metrics of the same classifiers in the nosia_all vs. the nosia_part settings (i.e., the original versus downsampling datasets), we find that downsampling cannot effectively address the severe data imbalance issue in the malware datasets. In fact, among the 16 pairs of comparison (4 classifiers by 4 datasets), downsampling leads to poorer performance in 13 cases (with the F1-score differences about 0.039 ± 0.002). Only in two cases (RF on M-3 and M-3s), downsampling improves the classifier performance. In Figure 4, we can see that this is because downsampling significantly improve the RF's prediction performance on the small families in M-3 and M-3s. We will further elaborate the impact of downsampling on different-scales malware families in the next RQ.

> *For the Android malware family classification task, a variety of multi-class classifiers can achieve very good overall prediction performance. Although downsampling can create more balanced data across families, it generally leads to worse prediction performance than using the original imbalanced data.*

***RQ2*: Do malware family classifiers obtained through normal machine learning methods perform consistently across large, medium and small families? How does downsampling affect the prediction result on different-scales of families?**

Figure 4 visualizes in box plots the performance of the four classifiers on the large/medium/small families in the four datasets. It shows the weighted average F1-score of a classifier overall the families of a particular scale in a dataset. As shown in Figure 4, it illustrates the prediction performance of normal machine learning methods on large, medium and small families in the four datasets. We can see a clear downward trend from the large families to the medium and small families for all 16 classifier-dataset combinations. Although the F1-score on the medium families are still acceptable (higher than 0.8) in 9 out of 16 cases in the nosia_all setting, the F1-scores on the small families of M-2/M-3/M-3s drop below 0.7 in 10 out of 12 cases in the nosia_all setting, and the F1-scores of the four classifiers on the small families of M-3s drop to around 0.5. The

[3]The tables of the performance metrics on the other three datasets are available at https://github.com/qWe1aSd/malPre.

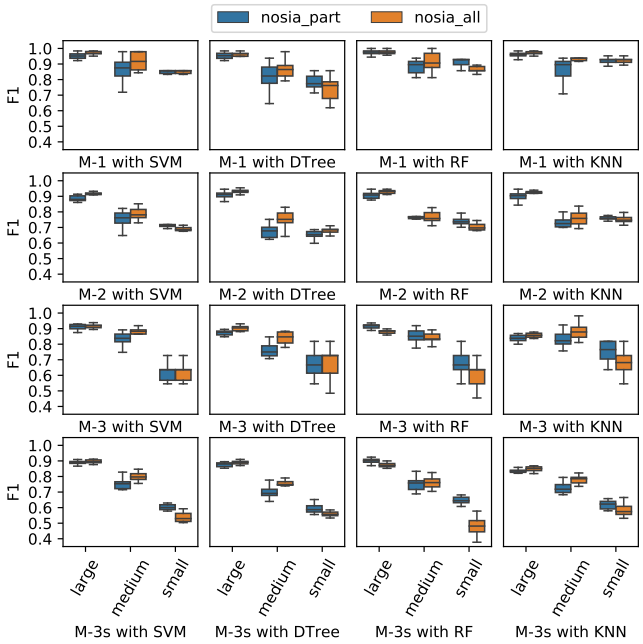

Figure 4: Performance by family scales: nosia_all versus nosia_part. All sub-figures share the same x-axis and y-axis.

relative better F1-score on the small families of M-1 is because M-1 has much fewer number of malware families (only 33) to predict than the other three datasets.

Figure 4 shows that the F1-scores on the large families in the nosia_all setting versus in the nosia_part setting are very close (with differences less than 0.035) for all 16 classifier-dataset combinations. That is, downsampling does not significantly affect the performance on the large families. However, the F1-scores on the medium families in the nosia_part setting are worse than those in the nosia_all setting in 15 out of 16 cases. For the small families, the F1-scores in the nosia_part setting are better than those in the nosia_all setting in 9 cases, close with differences less than 0.035 in 5 cases, and just worse in 2 cases. In the downsampling datasets, small families are less overwhelmed by the large and medium families. This helps the model capture important features of small families, and results in the performance improvements on the small families.

In Figure 4, we also observe that the variations of the F1-scores on the large families are small (evident in the narrow boxes and quantile marks) across the 10-fold cross validation for all 16 classifier-dataset combinations. This suggests that the model generalizes well on the large families. However, the F1-scores on the medium and small families have much larger variations across the 10-fold cross validation, which indicates poor generalizability. This poor generalizability becomes even more obvious when looking into the F1-scores on each family across the 10-fold cross validation. Figure 5 shows the F1-score variations (upper sub-figure) and variances (lower sub-figure) on each family in the M-3 dataset by SVM across the 10-fold cross validation. We can see that F1-score variance in the nosia_all setting (orange bar in Figure 5) is small for large families with > 100 malware applications. But F1-score variance is > 0.3 for many medium and small families.

In fact, the F1-score variances in the nosia_all setting are larger than 0.3 for 125 families in the four datasets (M-1: 11, M-2: 38, M-3: 21, M-3s: 55), among which 67 are medium families (M-1: 5, M-2: 19, M-3: 16, M-3s: 27) and 58 are small families (M-1: 6, M-2: 19, M-3: 5, M-3s: 28). For these 58 small families, the lowest F1-score is 0 and the highest F1-score is 1. The F1-score variances in the nosia_part setting (blue bar in Figure 5) have the similar situations. That is, downsampling cannot reduce performance variances. In some cases, downsampling may even increase the F1-score variances (see Figure 5). Similar observations can be made for other classifier-dataset combinations[4].

> *For the Android malware family classification task, all classifiers performs inconsistently on all datasets, with the best performance on the large families, acceptable performance on the medium families, but much worse performance on the small families. Furthermore, the classifiers suffer from poor generalizability on the medium and small families. Downsampling cannot make the classifiers perform consistently across different family scales. It cannot improve the model generalizability either.*

5 SIAMESE NETWORK TO THE RESCUE

In this section, we propose a siamese-network based learning method to tackle the performance and generalization issues for the few shot malware-family prediction revealed in the last section. We compare the overall performance of the malware-family classifiers obtained through the normal model training method versus the siamese-network based learning method, and the performance of these classifiers on large, medium and small families respectively. Our results confirm the superior performance of the siamese-network based learning method over the normal model training method, especially for the few shot malware-family prediction, which further lead to the significant improvement of the overall performance. By comparing the performance of MLP-based feature extractors obtained with the normal model training method versus the siamese-network based learning, we confirm that the performance improvements come from the siamese-network based learning, rather than the adoption of MLP-based feature extractor.

5.1 Siamese-Network Based Learning Method

Figure 6 presents our framework for siamese-network based learning. It consists of two steps: step 1 - train a MLP-based feature extractor by siamese network, and step 2 - perform malware-family classification with the trained MLP-based feature extractor.

As shown in Figure 6, a siamese network consists of two identical, weight-sharing Multilayer Perceptron (MLP) networks. This MLP network takes as input a 250-dimensional one-hot feature vector (see Section 3.1) of a malware application and produce as output a 250-dimensional real-valued, neural feature vector for this application. Therefore, we call this MLP network as the MLP-based feature extractor for malware applications. Our MLP consists of three fully connected layers, which contain 1024, 512 and 256 hidden units (i.e., neuron), respectively, and use ReLU ($ReLU(x) = max(0, x)$) [Glorot et al. 2011] as non-linear activation function. Our MLP is able to learn the implicit relationships among syntactic features.

[4]Please find the charts for other classifier-dataset combinations like Figure 5 at https://github.com/qWe1aSd/malPre.

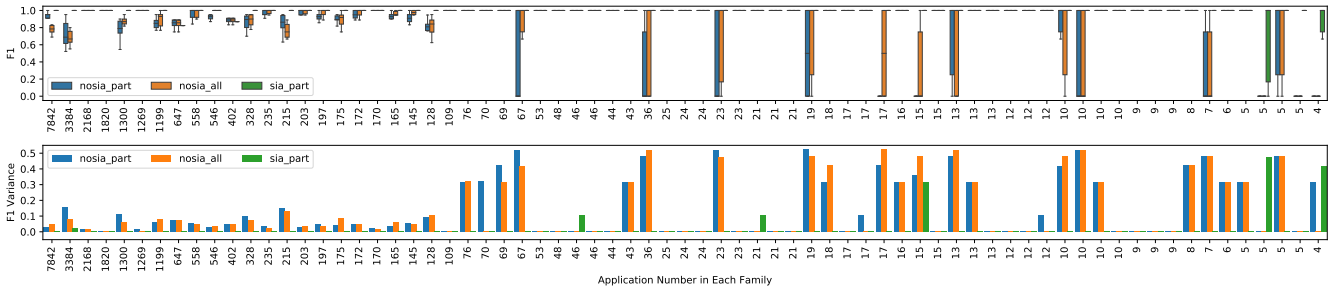

Figure 5: F1-score variations (upper sub-figure) and variance (lower sub-figure) for each family in M-3 by SVM across the 10-fold cross validation. Both sub-figures share the same x-axis.

Training the siamese network requires pairs of malware applications (one for each MLP network). Positive (or negative) pairs are the two malware applications from the same malware family (or different malware families). We create positive pairs pair-wisely in a malware family. If a family has N malware applications, we will create $N \times (N-1)/2$ positive pairs. We create negative pairs for each malware application in a family and all malware applications in the other families. Duplicate negative pairs will be discarded. Given a dataset for positive/negative pairs of malware applications, the training of the siamese network (i.e., the twin MLP networks) is guided by the contrastive loss function [Hadsell et al. 2006] as follows:

$$L(W, Y, X_1, X_2) = (1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2 \quad (1)$$

where D_W is the Euclidean distance, $\frac{1}{2}(D_W)^2$ is the partial loss function for a positive pair with $Y = 0$, $\frac{1}{2}\{max(0, m - D_W)\}^2$ is the partial loss function for a negative pair with $Y = 1$, and $m > 0$ is a margin to define a radius around each instance (X_1 or X_2) of pair. The target of contrastive loss function is to learn the W so that malware applications of the same families are pulled together and those of different families are pushed apart.

We create the dataset of positive/negative pairs of malware applications from the training data (see Section 4.1) of the downsampling ("part") dataset (see Section 3.3) of the M-1, M-2, M-3 and M-3s, respectively. Table 2 summarizes the number of positive/negative pairs created for the four datasets. Recall that the downsampling datasets have only 42.89%, 30.50%, 11.20% and 14.55% of the malware applications in the M-1, M-2, M-3 and M-3s datasets, respectively. However, the number of positive pairs created for a dataset is at least 4 times more than the number of malware applications in the original dataset. This creates more opportunities for the MLP-based feature extractor to learn the common features that the malware applications in the same family share, compared with learning the commonalities from each malware application individually in the normal model training method. Furthermore, the even more negative pairs create the opportunity to learn distinctive features to tell apart the malware applications from different families. This is especially important for the small families, as it helps the model distinguish the characteristics of the small families by contrasting their few instances with a large number of negative samples from other families. This explicit negative-contrast capability is absent in the normal model training method.

Table 2: The Number of Positive/Negative Pairs Created

	M-1	M-2	M-3	M-3s
Positive pair	16317	48834	118471	140562
Negative pair	126528	1316544	3639440	6169566

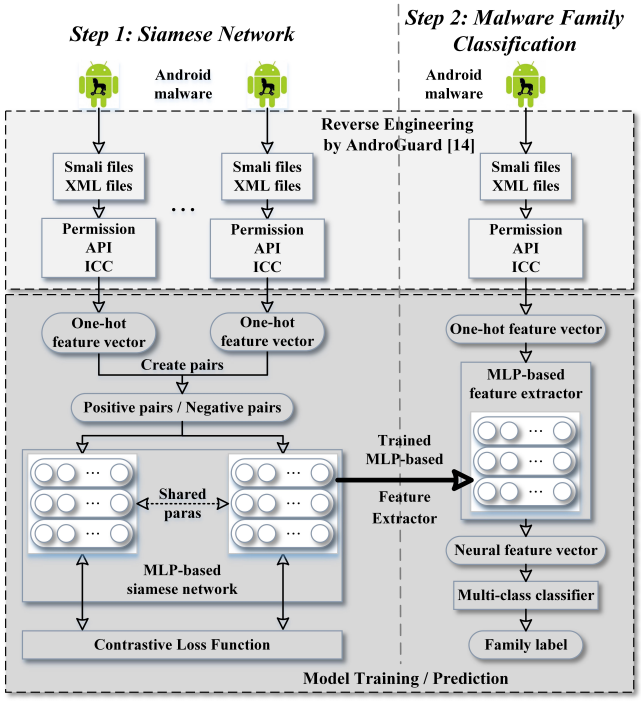

Figure 6: Siamese network based learning.

After training the siamese network, its MLP network is used as a feature extractor for malware family classification. The step 2 works in the same way as the normal multi-class classification in Figure 1. But the multi-class classifier in Step 2 takes as input the neural feature extractor outputted by the MLP-based feature extractor, rather than the one-hot syntactic feature vector. The training data is the 9 subsets of the "part" dataset. We refer to the classifiers obtained through siamese-network based learning as "sia_part". KNN can be directly applied to search the k-nearest neighbors by the neural feature vectors. SVM, DTree and RF are trained in the same way as described in Section 4.1. When training SVM/DTree/RF, we freeze the parameters of the MLP network.

5.2 Model Hyperparameter Optimization

Same as in Section 4.2, we use M-1 for hyperparameter optimization. Same hyperparameters are adopted for M-2, M-3 and M-3s. For the MLP networks in siamese network framework, we focus on **iteration**, **batch size** and **learning rate** as optimization targets. When tuning one of the three hyperparameter, the other two are fixed. **Iteration:** Generally, more iterations bring better accuracy rate during training neural network, but cost more training time. In order to balance the accuracy rate and time cost, we set the number of training iteration to 50 with the value space $\{1, 5, 10, 20, 50, 100, 150, 200, 500, 1000, 2000\}$. **Batch Size:** In neural network, batch size defines the number of malware applications whose classification loss will be backpropagated through the network. Based on a fitted batch size, neural network trains faster and achieve better results. In our experiments, we set 256 batch size for training siamese network after experimenting the batch size with 32, 64, 128 and 256. **Learning Rate.** The learning rate is the weight update rate of neural network in Adam algorithm [Kingma and Ba 2014]. The lower the learning rate is, the better capability of convergence the model obtains. We set a learning rates 0.0005 after experimenting this parameter with the value $\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$. We use the default value of the other hyperparameters in Adam according to keras documentation [Chollet et al. 2015]. The hyperparameters of the multi-class classifiers in Step 2 adopt the same settings as those described in Section 4.2.

5.3 Results

***RQ3*: How does the performance of the classifiers obtained by siamese-network based learning compare with that of the classifiers obtained by normal model training method?**

Figure 7 shows the overall performance of the four classifiers on the four datasets in both the sia_part and nosia_all settings. As sia_part and nosia_part uses the same "part" training data, and nosia_part generally has worse performance than nosia_all (see Figure 3), we show only nosia_all in Figure 7 for clarification.

We can see that the sia_part setting outperforms the nosia_all setting in all 16 classifier-dataset combinations, with 0.049–0.125 F1-score improvement. The lowest average F1-score in sia_part is 0.893 by DTree on M-2, but this F1-score is higher than the F1-scores of 12 out of 16 cases in nosia_all. Note that M-2 is the most challenging data as it has 131 families but only 5416 malware applications. The F1-scores of 15 out of 16 cases in sia_part are higher than 0.9, and the rest one has the F1-score 0.893. Furthermore, Figure 7 shows that the F1-score variations in sia_part become smaller than the F1-score variations in nosia_all for 13 out of 16 classifier-dataset combinations.

We conduct **T-test** [Paternoster et al. 1998] on the F1-score differences between the sia_part and nosia_all settings. Our T-test[5] shows that the F1-score differences between the two settings are statistically significant overall (p-value=$6.28E-50$) for all 160 runs (4 classifiers × 4 datasets × 10 runs), and are also statistically significant (p-value < 0.05) for the 10 runs of each classifier-dataset combination. Our T-test also shows that the differences in F1-score

[5]All test results are available at https://github.com/qWe1aSd/malPre.

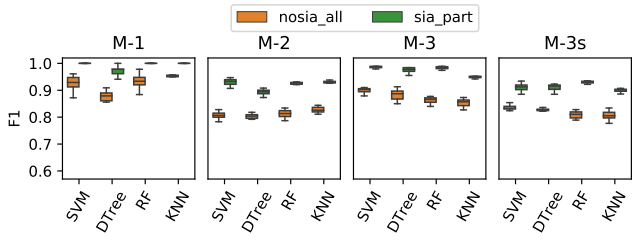

Figure 7: Overall performance: sia_part versus nosia_all. All sub-figures share the same y-axis.

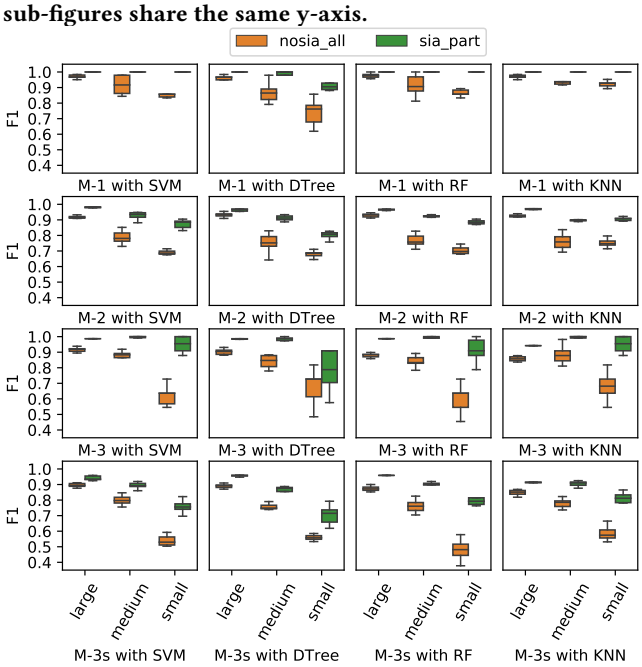

Figure 8: Performance by family scales: sia_part versus nosia_all. All sub-figures share same the x-axis and y-axis.

variance between the two settings are statistically significant (p-value=$4.26E-4$).

> *The classifiers obtained by siamese-network based learning perform statistically significantly better than those obtained by normal model training method, even though siamese-network based learning uses only a small percentage (e.g., 11.20% for M-3) of all data used in normal model training method. This performance boost cannot be achieved by simple data downsampling with normal model training method (see RQ1 in Section 4.3).*

***RQ4*: How does siamese-network based learning affect the performance on different-scales of families?**

Figure 8 show the performance of a classifier on the three family scales of a dataset in both the sia_part and nosia_all settings. We can see the performance boost by sia_part over nosia_all in all 48 cases (4 classifiers × 3 family scales × 4 datasets). The performance boost is the most evident for small families, with 0.077 – 0.348 improvement in F1-score for the 16 classifier-dataset combinations. Sia_part has relatively smaller performance boost for medium families, but the F1-scores on medium families still improve 0.058 – 0.159 for the 16 classifier-dataset combinations. The F1-scores on large families

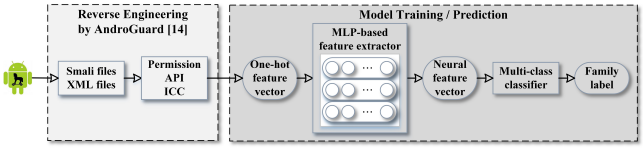

Figure 9: Malware family classification with MLP-based feature extractor trained by normal model training method.

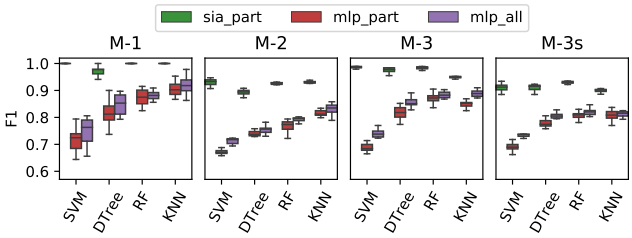

Figure 10: Overall performance: sia_part, mlp_part and mlp_all. All sub-figures share the same y-axis.

are already very high (around 0.91) in the nosia_all setting, but sia_part can still improve the performance further (with 0.029-0.107 F1-score improvement).

Figure 8 shows that accurate prediction on small families is still more challenging than that on medium and large families, However, the performance gap across the three family scales become much narrower in the sia_part setting than in the nosia_all setting. The F1-scores on the large/medium/small families of the M-1 dataset are almost the same (with the differences < 0.093). For the other three datasets, the F1-score differences between the large families and the small families in the sia_part setting are < 0.1 in 5 out of 12 classifier-dataset combinations, and are in $0.1 - 0.25$ in the other 5 cases. In contrast, the F1-score differences between the large families and the small families in the nosia_all setting are > 0.25 in 7 out of the 12 cases.

The green bars in Figure 5 show the F1-score variances for each family in M-3 by SVM across the 10-fold cross validation in the sia_part setting. sia_part has only 1 medium families and 2 small families with F1-score variance larger than 0.2. This is much fewer than nosia_all (16 medium families and 5 small families with F1-score variance larger than 0.2) and nosia_part (17 medium families and 6 small families with F1-score variance larger than 0.2). Furthermore, there are 65 families with 0 or < 0.02 F1-score variances across the 10-fold cross validation in sia_part. In contrast, nosia_all and nosia_part have only 32 and 27 such families respectively.

> *Siamese-network based learning can significantly boost the performance of malware family classification for all scales of families. The smaller the family is, the larger it can boost the performance. As such, it can narrow the performance gap between the large and small families. Combining downsampling with siamese-network based learning can improve the model generalizability, especially for medium and small families.*

***RQ5*: Can the malware family classification with MLP-based feature extractor trained by the normal model training methods achieve the same performance boost as the MLP trained by siamese-network based learning?**

Figure 9 shows the framework for the malware family classification task with MLP-based feature extractor trained by normal model training method. Different from the framework in Figure 1 in which the classifier takes as input the one-hot syntactic feature vector, the classifier in Figure 9 takes as input the neural feature vector computed by the MLP network from the one-hot syntactic feature vector. The MLP network in Figure 9 has the same network structure as the MLP network in Figure 6. At the inference time, the framework in Figure 9 works in the same way as the Step 2 in Figure 6, but the MLP network and the multi-class classifier are trained in different ways. In Figure 6, the MLP network is trained by the siamese network in Step 1 and its parameters are frozen when training the classifier in Step 2. In contrast, the MLP network in Figure 9 uses the normal model training method described in Section 4.1 with cross entropy loss function. Then, we freeze its parameters for training the later multi-class classifiers in Figure 9. Note that both MLP network and classifiers adopt the same hyper-parameters described in Section 4.2 and Section 5.2. The training is done for both the "all" and "part" datasets, which is referred to as mlp_all and mlp_part respectively.

Figure 10 shows the overall performance of a classifier on a dataset in the three different settings: sia_part, mlp_all and mlp_part. We can see that sia_part achieves significantly higher ($0.06 - 0.299$) F1-scores than mlp_all and mlp_part in all 16 classifier-dataset combinations. The improvement is statistically significant overall for all 160 runs (4 classifiers $\times$ 4 datasets $\times$ 10 runs), and the 10 runs of each classifier-dataset combination. In fact, the performance of mlp_all is even worse than that of nosia_all in 14 out of 16 classifier-dataset combinations (see Figure 3). For mlp_part versus nosia_part, the number of such cases is 12 out of 16 cases. Recall that nosia_part still has two cases (RF on M-3 and M-3s) in which downsampling leads to 0.033 and 0.034 F1-score improvement over nosia_all (see Figure 3). Unfortunately, the performance of mlp_part is either close to or worse than that of mlp_all.

An interesting observation is that KNN is the best performer over the SVM, DTree and RF on all the four datasets in both mlp_all and mlp_part. The performance gap between the KNN and the other three classifiers is generally larger on the small dataset M-1 and the medium dataset M-2, compared with the performance gap on the two large datasets M-3 and M-3s. Note that KNN is a non-parametric method and does not training, while the other three classifiers require training. In contrast, the performance gap of the four classifiers in sia_part is much narrower (F1-score difference < 0.05), no matter the classifiers require training or not.

The results in Figure 10 indicates that the MLP network is not well trained by the normal model training method. As the MLP network involves three fully connected layers, training the classifier with the MLP network would require more data than training the classifier alone. The size of malware family datasets does not satisfy this requirement. As a result, the MLP trained by normal model training method performs poorly in encoding the one-hot syntactic feature vector into the neural feature vector, which turns out to be even harder to classify. In contrast, we can prepare a large number of positive/negative pairs for sufficiently training the MLP network in siamese network. This well-trained MLP network can

effectively extract neural features to distinguish malware applications in the same or different families, which leads to significantly better classification performance.

> *MLP-based feature extractor, if not well trained by siamese network, cannot result in the performance boost in malware family classification, but the performance degradation.*

6 THREATS TO VALIDITY

The study presented in this paper is subject to some limitations that could potentially threaten our experiment results and findings.

An internal threat is the hand-picked syntactic features used to represent malware applications. We do not use all permissions, APIs and ICC attributes in Android API documentation. Instead, we use a subset of these syntactic features, which has been shown to be effective to detect or classify Android malware in several studies [Avdiienko et al. 2015; Deshotels et al. 2014; Wang et al. 2014]. Meanwhile, those syntactic features are interpretable because of a detailed description in Android documentation, and can be conveniently extracted from source code without the need for complex program graph analysis.

Another internal threat is the downsampling strategy and the definition of the family scale. Due to the severe data imbalance (thousands malware applications in some families but just a few in others), we cannot use a simple downsampling approach to downsample all families to the smallest family size. Therefore, we adopt the stratified sampling method by different family scale. Based on the observation of the family size distribution, we empirically set the families with more than 100 malware applications as large, those with 10 to 100 malware applications as medium, and those with less than malware applications as small. This family scale definition allows us to compare the performance across the datasets. For a dataset with significantly different family size distribution, the family scale boundary may have to be refined, which may affect some experiment results. However, it should not fundamentally affect our conclusions, especially those for small families.

An external threat is the quality and representativeness of the malware family datasets we use in this study. These four datasets have been constructed by security experts, and have also been used in existing studies for malware analysis. We believe the quality of malware families in these datasets should be good. Furthermore, these four datasets cover a wide range of malware families that appear over a long period of time. Thus, the malware families in these datasets should be representative. And we observe very similar data characteristics and obtain very similar results across the four datasets, which also indicates the generalization of our findings. Of course, we can further confirm our finding on new malware family dataset when they become available.

7 CONCLUSION AND FUTURE WORK

In this paper, we zoom into the problem of malware family classification at three different family scales (large, medium and small). We find that although the multi-class malware family classifiers achieve very good classification accuracy at the overall level and for the large families, the accuracy drops significantly for few shot malware families. Furthermore, the generalizability of the classifiers also become worse for few shot malware families. Our findings are generalizable for the four malware family datasets with very different sets of malware applications and very different definition of malware families, as well as very different number of malware applications and malware families.

Motivated by our findings, we propose a siamese-network based learning method and a MLP-based classifier for addressing the issue of few shot malware family classification. Our method is completely different from existing methods in terms of feature learning and model training. Our experiments show that the performance of malware family classifiers obtained through our method are significantly better than that of the classifiers obtained through normal model training method in existing work, even though our classifiers are trained using only a small percentage of the whole dataset used in normal model training method. Furthermore, our classifiers narrow the performance gap between the large and few shot families and improves the model generalizability, especially for few shot families. We also show that these benefits cannot be obtained through simple downsampling method or just MLP-based feature extractor, without using the siamese-network based learning method.

In the future, we are interested in experimenting different syntactic features for the malware family classification task, and exploring external knowledge (e.g., API description embeddings) to address the few shot malware family classification issue.

ACKNOWLEDGMENTS

We appreciate the anonymous reviewers for their beneficial feedback. This work has partially been sponsored by the National Science Foundation of China (No. 61872262, 61572349).

REFERENCES

Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Yves Le Traon, et al. 2016. Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211.

Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Acm Sigplan Notices*, Vol. 49. ACM, 259–269.

Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 217–228.

Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining apps for abnormal usage of sensitive data. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 426–436.

Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. 2016. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*. Springer, 850–865.

Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a "siamese" time delay neural network. In *Advances in neural information processing systems*. 737–744.

Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter* 6, 1 (2004), 1–6.

Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *computers & security* 73 (2018), 326–344.

Sen Chen, Minhui Xue, Zhushou Tang, Lihua Xu, and Haojin Zhu. 2016. Stormdroid: A streaminglized machine learning-based system for detecting android malware. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications*

Security. ACM, 377–388.
François Chollet et al. 2015. Keras. https://keras.io.
Ronan Collobert and Samy Bengio. 2004. Links between perceptrons, MLPs and SVMs. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 23.
Luke Deshotels, Vivek Notani, and Arun Lakhotia. 2014. Droidlegacy: Automated familial classification of android malware. In *Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop 2014*. ACM, 3.
Anthony Desnos et al. 2011. Androguard.
Amr ElRafey and Janusz Wojtusiak. 2017. Recent advances in scaling-down sampling methods in machine learning. *Wiley Interdisciplinary Reviews: Computational Statistics* 9, 6 (2017), e1414.
Lingling Fan, Ting Su, Sen Chen, Guozhu Meng, Yang Liu, Lihua Xu, Geguang Pu, and Zhendong Su. 2018c. Large-scale Analysis of Framework-specific Exceptions in Android Apps. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, 408–419.
Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. 2018b. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security* 13, 8 (2018), 1890–1905.
Yaqing Fan, Xinya Cao, Jing Xu, Sihan Xu, and Hongji Yang. 2018a. High-Frequency Keywords to Predict Defects for Android Applications. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. IEEE, 442–447.
Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 576–587.
Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. 2016. Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv preprint arXiv:1608.06254* (2016).
Sa Gao, Chunyang Chen, Zhenchang Xing, Yukun Ma, Wen Song, and Shang-Wei Lin. 2019. A neural model for method name generation from functional description. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 414–421.
Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, obfuscation-resilient detection and family identification of android malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 3 (2018), 1–29.
Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 315–323.
Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. 2019. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 810–822.
Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR '06)*. IEEE Computer Society, 1735–1742.
Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 125–136.
Haibo He and Edwardo A Garcia. 2008. Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering* 9 (2008), 1263–1284.
Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks* 13, 2 (2002), 415–425.
Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).
Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2.
Ruchika Malhotra. 2016. An empirical framework for defect prediction using machine learning techniques with Android software. *Applied Soft Computing* 49 (2016), 1034–1050.
McAfee-Labs. 2018. *McAfee Labs Threats Report*. Retrieved August, 2019 from https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf
Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. 2017. Deep android malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 301–308.
Eugene Ndiaye, Tam Le, Olivier Fercoq, Joseph Salmon, and Ichiro Takeuchi. 2018. Safe Grid Search with Optimal Complexity. *arXiv preprint arXiv:1810.05471* (2018).
Nokia. 2019. *Nokia Threat Intelligence Report*. Retrieved August, 2019 from https://onestore.nokia.com/asset/205835?did=d0000000016z&utm_campaign=threatintelligence18&utm_source=marketo&utm_medium=LandingPage&utm_content=report&utm_term=awareness
Raymond Paternoster, Robert Brame, Paul Mazerolle, and Alex Piquero. 1998. Using the correct statistical test for the equality of regression coefficients. *Criminology* 36, 4 (1998), 859–866.
Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 729–746.
Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. 1995. Nearest neighbor queries. In *ACM sigmod record*, Vol. 24. ACM, 71–79.
S Rasoul Safavian and David Landgrebe. 1991. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics* 21, 3 (1991), 660–674.
Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*. 4077–4087.
Stephen V. Stehman. 1997. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment* 62, 1 (1997), 77–89.
Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Jorge Blasco. 2014. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications* 41, 4 (2014), 1104–1117.
Aixin Sun, Ee-Peng Lim, and Ying Liu. 2009. On strategies for imbalanced text classification using SVM: A comparative study. *Decision Support Systems* 48, 1 (2009), 191–201.
Chongbin Tang, Sen Chen, Lingling Fan, Lihua Xu, Yang Liu, Zhushou Tang, and Liang Dou. 2019. A Large-scale Empirical Study on Industrial Fake Apps. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '10)*. IEEE Press, 183–192.
Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. 2004. A primer on kernel methods. *Kernel methods in computational biology* 47 (2004), 35–70.
Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 297–308.
Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. 2014. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security* 9, 11 (2014), 1869–1882.
Yaqing Wang and Quanming Yao. 2019. Few-shot learning: A survey. *arXiv preprint arXiv:1904.05046* (2019).
Yu-Xiong Wang and Martial Hebert. 2016. Learning to learn: Model regression networks for easy small sample learning. In *European Conference on Computer Vision*. Springer, 616–634.
Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.
Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. 2016. Hydra: Massively compositional model for cross-project defect prediction. *IEEE Transactions on software Engineering* 42, 10 (2016), 977–998.
Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 51–62.
Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 371–372.
Sergey Zagoruyko and Nikos Komodakis. 2015. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4353–4361.
Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 1105–1116.
Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. 2013. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 611–622.
Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*. IEEE, 95–109.