The Thirty-First International Florida
Artificial Intelligence Research Society Conference (FLAIRS-31)

Learning to Identify Known and Unknown Classes: A Case Study in Open World Malware Classification

Mehadi Hassen, Philip K. Chan
School of Computing, Florida Institute of Technology
Melbourne, USA
mhassen2005@my.fit.edu
pkc@cs.fit.edu

Abstract

In this paper we propose an open world malware classification. Our approach is not only able to identify known families of malware but is also able to distinguish them from malware families that were never seen before. Our proposed approach is more accurate and scales better on two evaluation datasets when compared to existing algorithms.

Introduction

In a classification problem, we are given a set of classes between which the model has to learn to discriminate. There are many cases in which we know the possible classes in the problem domain beforehand and simply learning to distinguish between these classes is sufficient. This can be thought of as the "closed world" scenario. In other problem domains, however, we are aware of only some number of classes during training and instances that belong to classes not present in the training set can be present in a test set. In this paper we will refer to this as the "open world" scenario.

This is especially true in the domain of malware family classification, in which we want to identify the family of a malware sample. There are a variety of reasons why this is the case. One reason is that it's not possible to collect all samples from every existing malware class/family for training. Even if we were able to do so, because of the adversarial nature of this problem domain, malware authors constantly release new malware families. Therefore, it is important to have a classifier that is not only capable of discriminating between instances from known malware families but one that is also able to determine if samples do not belong to any of the known families and label them as $unknown$. In this paper we present a classifier system that achieves this goal. Our proposed approaches combine a classifier and an outlier detector to build a system where the outlier detector is trained on new features extracted from the output of the classifier. The following points summarize our main contributions:

- We present a different look from the majority of the research in the malware classification domain by addressing classification in an open world.
- We propose the extraction of new features from classifier output which transforms the problem of identifying unknown class instances into a new feature space.
- We provide an approach that is more accurate and scales better than previous open world classification approaches on the evaluation datasets.

Related Works

To the best of our knowledge there are very few works in the malware classification domain that focus on having the capability to operate in an open world scenario. One such work in the malware domain comes from K. Rieck et al.(Rieck et al. 2011) in which malware samples are clustered into families and then the distance of a test instance to the closest cluster centroids is used as an outlier score. The limitation of this approach comes from the use of unsupervised methods for identifying between the known families which does not take advantage of label information that might be present for the known classes.

Open world classification has aspects similar to anomaly detection problems. Anomaly detection is a well researched area and has found application in the security realm such as intrusion detection. For example to mention a few, Ramaswamy et al. (Ramaswamy, Rastogi, and Shim 2000) propose a distance based approach which looks at the distance of an instance from its k^{th} nearest neighbor to identify outliers. Breunig et al. (Breunig et al. 2000) propose a density based approach that looks at the local density of a point to determine its anomaly score. Many more approaches have been proposed for this problem; we direct the reader to (Chandola, Banerjee, and Kumar 2009; Tan, Steinbach, and Kumar 2006; Zhang 2013) for a more detailed survey of the different anomaly detection techniques. The similarity of anomaly detection to open world problem is that in both cases there is an interest in identifying outlier samples. However, there are two main differences. First, the learning task in anomaly detection is in only detecting anomalous samples and not learning to discriminate the normal samples into different classes. Second, the assumption which is taken in anomaly detection is that outlier instances are rare (Tan, Steinbach, and Kumar 2006). This assumption, however, does not hold in the case of the open world problem.

Scheirer et al. (Scheirer et al. 2013) propose a modification to one-class SVM, called 1-vs-set SVM, to handle a one class open world problem for image recognition. Multi-class anomaly detection without classification called Kernel Null Foley-Sammon Transform (KNFST) proposed in (Bodesheim et al. 2013) shares one of the objectives of the open world problem, which is to identify instances that do not belong to any of the known classes. Bodesheim et al. (Bodesheim et al. 2015) build on this idea by taking a local approach where they only considering k closest elements to an instance when deciding the novelty on a test instances. In (Jain, Scheirer, and Boult 2014), Jain et al. present a multi-class open set classifier framework PI_SVM for fitting a single-class probability model over the known class scores from a discriminative binary classifiers.

Approach

The prediction of a classifier can be obtained in the form of predicted class probabilities $Pr(y_i = c \mid \vec{x_i})$, where $c \in C^k$ is a class label among the known classes in a training set, $\vec{x_i}$ is the feature vector of instance i, and y_i is the class label of instance i. Each of these probabilities can be interpreted as the confidence of the classifier labeling an instance as belonging to that class.

Our approaches are based on the intuition that class probabilities predicted by a classifier can help distinguish unknown class instances from known class instances. We expect the classifier to be less confident when making a prediction about instances from an unknown class as compared to instances from the known class.

New Derived Features

We propose extracting features from the output of a multi-class classifier to help identify known class instances from unknown class instances. The first feature we extract is $P_{max} = \max_{c \in C^k} Pr(y_i = c \mid \vec{x_i})$. We aim to capture the confidence of the classifier by extracting the P_{max}.

In addition to P_{max} we also want to capture another perspective on the prediction confidence by observing how spread out the predicted class probabilities are. For example assume a classifier trained to classify between 3 classes makes prediction for instance A with probabilities of 0.8, 0.1 and 0.1 for classes c1, c2 and c3 respectively. Also assume that for instance B it makes prediction with probabilities 0.8, 0.2 and 0.0. Although their maximum prediction probabilities for both example instances are the same at 0.8, the fact that the classifier predicts A as possibly belonging to all three classes, albeit with low probabilities for class c2 and c3, gives more insight about the classifiers confidence on its prediction. We try to capture this by measuring the entropy of the predicted class probabilities. Entropy quantifies the diversity in the predicted class probabilities(in other words it tells us how unevenly distributed the predicted class probabilities are.) The entropy for probability distribution p over $|C^k|$ classes is defined as $entropy(p) = -\sum_j^{|C^k|} p_j \log p_j$.

Algorithm 1 outlines the procedure used to extract new features from predicted class probabilities. The algorithm starts by splits the training set into T segments (line 1).

Algorithm 1: Use T-fold cross validation to generate an outlier detector training set with the new features.

Input :
D: Training set consisting of feature matrix X and class labels Y. The class labels are from the known class C^k.
Output:
X_{inlier}: Training data represented in terms of new prediction probability based features.

1 Split D into T segments;
2 Initialize X_{inlier} as empty;
3 **for** $t = 1$ *to* T **do**
4 Train multi-class classifier model M_{tmp} on D without D_t;
5 $P_{inlier} \leftarrow$ Predict class probabilities for D_t using M_{tmp};
6 $X_{tmp} \leftarrow$ extract features from P_{inlier};
7 $X_{inlier} \leftarrow X_{inlier} \cup X_{tmp}$;
8 return X_{inlier};

For each segment, first a classifier is trained on data D excluding the instances in D_t (line 4). Then, predictions are made on the instances in D_t. Afterwards, two features are extracted from the predicted class probabilities: the maximum predicted class probability P_{max} and the $entropy$ of the predicted class probabilities. Finally, the extracted features X_{tmp} for each instance in D_t are added to X_{inlier}, which holds the representation of the training set in terms of the new features.

Classification in an Open World (COW)

By considering different aspects of a classifiers output, such as P_{max} and $entropy$, we should be able to distinguish between instance belonging to known classes C^k from those belonging to unknown classes C^u. One of the challenges of trying to recognize instances from classes in C^u comes from the fact that we can only get training data representing classes in C^k. In our case this means that during training we can get only the predicted class probabilities for instances from classes in C^k. To address this, we use an outlier detection method trained on predictions of the instances from C^k.

Our proposed approach uses a classifier and an outlier detector to build an open world classification system (COW). Algorithm 2 outlines the procedure for training COW. The algorithm starts by calling Algorithm 1 to extract the new features which will be used to train the outlier detector in COW. An outlier detector $M_{outlier}$ is then trained using X_{inlier} (line 2). Finally, a multi-class classifier is trained on the original dataset D, and the two models are returned (lines 3-4). During testing, the multi-class classifier M_{multi} is first used to make predictions about a test instance. Then the outlier detector $M_{outlier}$ is used to determine if the instance is an inlier in which case the predicted class label is used. Otherwise it is labeled $unknown$.

Algorithm 2: Training COW

Input :
D: Training set consisting of feature matrix X and class labels Y. The class labels are from the known classes C^k.
Output:
M_{multi}: Multi-class classifier model
$M_{outlier}$: Outlier detection model

1 $X_{inlier} \leftarrow$ `Algorithm1` (D) ;
2 Train outlier detector model $M_{outlier}$ on X_{inlier};
3 Train multi-class classifier model M_{multi} on D;
4 return M_{multi} and $M_{outlier}$;

Algorithm 3: Training COW_PC

Input :
D: Training set consisting of feature matrix X and class labels Y. The class labels are from the known class C^k.
Output:
M_{multi}: Multi-class classifier model
$ListM_{outlier}$: List of outlier detection model

1 $X_{inlier} \leftarrow$ `Algorithm1` (D) ;
2 Initialize $ListM_{outlier}$ as empty;
3 **foreach** *class* `c` *in* C^k **do**
4 $X_{inlier\_c} \leftarrow X_{inlier}$ rows corresponding to instances correctly predicted as `c`;
5 Train outlier detector model $M_{outlier\_c}$ using X_{inlier_c};
6 Add $M_{outlier\_c}$ to $ListM_{outlier}$;
7 Train multi-class classifier model M_{multi} on D;
8 return M_{multi} and $ListM_{outlier}$;

Per Class Classification in an Open World (COW_PC) COW trains one global outlier detection model for all the known classes. This, however, might face challenges in scenarios in which a classifier is not equally good in identifying all the known classes. In such a case the classifier might make predictions about certain classes with ease while struggling for other classes. So predictions made about the difficult classes might be confused for predictions of instances from an unknown class. In trying to address such a scenario we present a modified version of COW which we call COW_PC. COW_PC trains separate outlier detection models per each class in C^k. By doing so we aim to address the aforementioned challenges.

The training procedure for COW_PC, Algorithm 3, is a modified version of Algorithm 2. Similar to COW, new features are extracted (line 1) and a classifier M_{multi} is trained using the original dataset (line 7). The difference between the two approaches lies in training of the outlier detector (lines 2-6). In case of COW_PC separate outlier detection models, one for each class in C^k, are trained and then added to the list of outlier detectors.

During testing, similar to COW, the multi-class classifier M_{multi} is first used to make a prediction. In the case of COW_PC, if the instance is predicted as class $c \in C^k$ then the outlier detector for class c ($M_{outlier_c}$) is then used to determine if the instance is an inlier in which case the predicted class label is used. Otherwise it is labeled $unknown$.

Experimental Evaluation

Evaluation Datasets

We used two datasets for evaluating the proposed approaches. The first is the Microsoft Malware Classification Challenge (MS-Challenge) dataset (msc 2015). This dataset contains 10,867 disassembled and labeled windows malware binaries from 9 malware families/classes. Our disassembled file parser was able to properly parse 10,260 of the samples used in the following experiments.

We also evaluate on the Android Malware Genome Project (mal) dataset which contains different families of malicious android apps. The original dataset contained some classes with very few samples. This presented a challenge when performing the evaluation because we setup the open world experiments resulting in too few training samples. For this reason we use only those classes which have at least 40 samples. This results in a dataset that consists of 986 samples from 9 classes.

Simulating Open World Scenario

To simulate an open world scenario we first take a dataset and randomly designate $|C^u|$ number of classes to constitute C^u and the remaining classes constitute C^k. We add all the instances belonging to classes in C^u to the test set. We randomly added 75% of instances belonging to classes in C^k to the training set and the remaining 25% to the test set.

Malware Features and Learning Algorithms

The malware features we used to train our classifier models in these experiments are based on the research of Hassen and Chan (Hassen and Chan 2017). These are features extracted from the function call graph (FCG) of windows and android applications. During the extraction of these features, the functions in the FCG are first clustered using Locality Sensitive Hashing (LSH) and the resulting cluster ids are used to label the functions. Then a vector representation of the FCG is created. Even though the original paper uses a two-level classifier system with these features, we use a single classifier in our experiments.

All the experiments involving COW and COW_PC were preformed with Random Forest (RF) (Breiman 2001) as the classifier model and KD-Tree based Kernel Density Estimation as the outlier detector.

Discriminating Known Class Instances from Unknown Class Instances

To evaluate the ability to discriminate known class instances from unknown class instances, we use Area Under Curve of the ROC as a measurement metric. We use the outlier score and compute the True Positive Rate and False Positive Rate. When generating these ROC figures, we treat identifying the

Table 1: TPR at a very low FPR of 0% for all four methods on both datasets.

	TPR at 0% FPR	
	MS-Challenge Dataset	Android dataset
COW	42.41%	81.21%
COW_PC	69.81%	76.90%
PI_SVM	23.97%	72.11%
KNFST	9.94%	77.25%

known classes as positive class and identifying the unknown class as the negative class. We chose to present the results algorithm in this manner because the main objective of the system is in classifying malware. Therefore, doing so in an open world scenario involves first identifying if an instance is indeed from a known class.

We compare our approaches with two other previous approaches on open set recognition: PI_SVM (Jain, Scheirer, and Boult 2014; lib 2017) and KNSFT (Bodesheim et al. 2013; knf 2017). For the sake of completeness we also report the performance of using an outlier detector, trained on the original data input space rather than on the prediction probability features we recommend in this paper, to identify unknown class instances. The results for this outlier detector are reported under the name "Original Input" in figures 1a and 1b. For PI_SVM and KNFST we use the original authors implementation of the algorithms.

We performed 10 experiments for each approach. In each experiment the training and test sets are created to simulate an open world scenario by setting the number of unknown classes $|C^u|$ to be 3. We then use the learned model to generate an outlier score for each test instance to indicate the degree to which the model believes the instance does not belong to any of the classes in C^k. Figure 1a shows the result of these experiments carried out on the MS-Challenge dataset in the form of the average ROC. Figure 1b, on the other hand, shows the result of similar experiments on the Android Malware Genome Project dataset.

We would like to highlight two observations from the results in Figures 1a and 1b. First, our proposed approaches perform better compared to PI_SVM and KNFST. For example on the MS-Challenge dataset, at a 10% false positive rate (i.e. where 10% of instances from unknown classes get predicted as belonging to one of the known classes) our two approaches COW and COW_PC achieve a true positive rate (i.e. percentage of instances from known classes that are detected as known) of 95.36% and 95.61% respectively. This results in an area under the curve(AUC) up to 10% FPR of 0.0917 and 0.0923. At the same point the PI_SVM and KNFST achieve TPR of 82.76% 81.89%, respectively.

Second, our approach achieves a relatively high TPR even at 0% FPR. Table 1 shows, for instance, in case of MS-Challenge dataset our approach COW_PC achieves 69.81% TPR compared to PI_SVM's 23.97%. Similarly, for the Android dataset COW achieves 81.21% TPR compared to 77.25% TPR of KNFST.

Table 2: Weighted average precision, recall and f-score.

	MS-Challenge Dataset		
	Precision	Recall	F-score
COW	0.94	0.90	0.91
COW_PC	0.92	0.90	0.90
PI_SVM	0.94	0.80	0.85
	Android Dataset		
	Precision	Recall	F-score
COW	0.95	0.86	0.89
COW_PC	0.93	0.84	0.87
PI_SVM	1	0.66	0.78

Table 3: Comparing the training and test times. The MS dataset has average training and test size of 5151 and 5110, respectively. The android dataset has training size of 480 and test size of 506 on average.

Algorithm	MS Dataset Average Time (sec)		Android Dataset Average Time (sec)	
	Training	Test	Training	Test
COW	9.15	2.11	4.11	0.24
COW_PC	7.42	1.42	4.06	0.41
PI_SVM	79.06	34.87	3.41	1.90
KNFST	577.50	202.10	3.96	2.21

Discriminating Among Known Classes

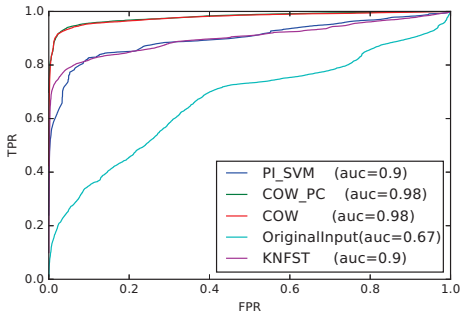
The results presented so far show how well the different approaches perform on the task of distinguishing between known class instances and unknown class instances. In addition to this, an open world classifier also needs to be able to discriminate between the known classes.

To evaluate this we run experiments on training and test sets created to simulate an open world scenario. In each experiment we set the number of unknown classes $|C^u|$ to be 3, which means $|C^k|$ will be 6 for both datasets. To make sure that all possible $\binom{|C^k|+|C^u|}{|C^u|} = \binom{9}{3}$ combinations of classes are used in C^u, 84 such experiments are performed. Hence, each class in the two datasets gets to be in the known class set exactly 56 times. In each experiment we recorded the weighted average precision, recall and f-score values of each class in the known class. This is obtained by first calculating the average of these three metrics for each known class. Then further calculating the average across all the known classes while weighting the values of these metrics by the fraction of the size of each known class in the test sets.

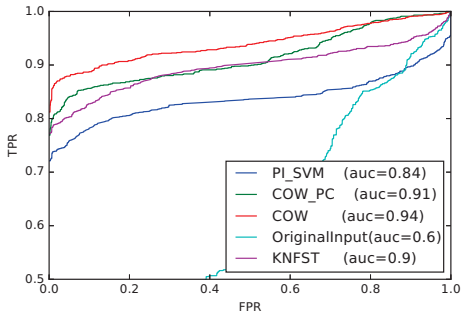
Since KNFST does not have the capability to discriminate between the known classes, we report results for COW, COW_PC, and PI_SVM Table 2. All three algorithms have one hyperparameter that specifies the threshold for discriminating between known and unknown class instances. We propose using a validation set to perform binary search over the hyperparameter space using f-score as the search metrics.

Efficiency Comparison with other Approaches

To compare the time efficiency of our approaches with PI_SVM and KNFST, we record the training and test times when running the experiments. Table 3 presents the average



(a) Average ROC up to 100% FPR on MS Dataset



(b) Average ROC up to 100% FPR on Andriod Dataset

Figure 1: Average ROC from 10 runs for distinguishing between instances from known and unknown classes.

training and test time of 10 runs on MS and android datasets. These experiments were carried out on a machine with an Intel-i7 2.60GHz processor and 20GB RAM. As the results in Table 3 show, our two approaches, COW and COW_PC, seem to scale better compared to the other two algorithms.

Effect of Number of Known Classes

Another interesting phenomenon to study is how our approaches perform as the number of known classes varies while keeping the number of unknown classes constant. The purpose of these experiments is to try to understand whether gathering more malware families for training can help improve performance of the open world classifiers.

We set up the experiment in the following manner for both of the evaluation datasets. First, we select the original test and training data to simulate an open world scenario. Afterwards, we chose t number of classes at random from the known classes C^k in the training set, and create a new training set from the previously created training set that contains instances from the t selected classes. As for the test dataset we select all the instances from the unknown class that are in the previously created test set together with the instances from the t selected known classes. We then train a model on the new training set and evaluate it on the new test set. We record the performance of the model in terms of the AUC up to a FPR of 10%.

Another way to understand these experiments is in terms of the percentage of opennes defined by (Scheirer et al. 2013), Equation 1. For a closed world scenario where the same classes are seen both during training and testing, we get an openness value of 0. On the other hand a higher openness value indicates a more open problem. In the context of these experiments, increasing the number of known classes during training while keeping the number of unknown classes seen during testing constant will result in decreasing openness. We expect that this decrease in openness should in turn result in improved performance. This expectation aligns with what Jain et al. (Jain, Scheirer, and Boult 2014) show in their experiments.

$$openness = 1 - \sqrt{\frac{2 \times |\ training\ classes\ |}{|\ test\ classes\ | + |\ target\ classes\ |}} \quad (1)$$

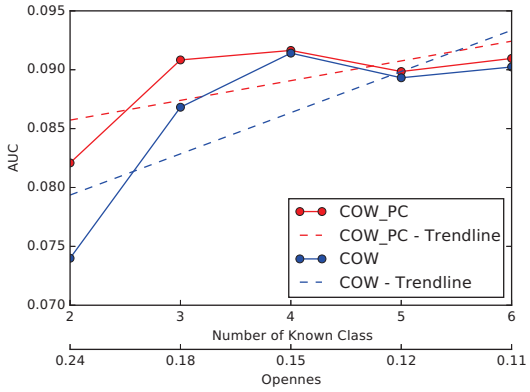
The result for MS-Challenge dataset indeed agrees with the expectation that as the openness decreases (i.e. number of known classes increases) the performance of our approach improves, Figure 2a. The more surprising result comes in the case of the Android dataset, Figure 2b. In this case we see that performance actually degrades as openness decreases (i.e. as number of known classes increases).

We hypothesize that the unexpected results have to do with the number of instances in each class. When we compare the class distributions of the MS-challenge dataset with the Android dataset, we see that the number of instances per class in the android dataset is considerably smaller. Generally, the classification task becomes more difficult as the number of classes to classify increases and also as the number of training instances gets smaller. Since our approach depends on the predicted class probabilities generated by the classifier, we believe the smaller size of the Android dataset has resulted in decreased performance as the number of classes increases (openness decreases). This hypothesis also helps explain why our approach records lower AUC in case of the android dataset as presented in previous Sections.

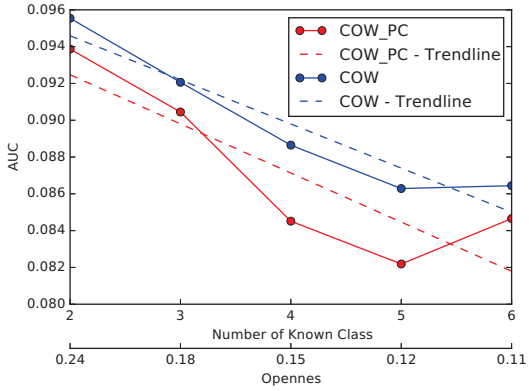
We evaluated this hypothesis by down-sampling the MS-Challenge dataset to have a comparable number of instances for each class with the Android dataset. The results in Figure 3 show the average AUC up to a 10% FPR of 10 runs. We observe that the AUC generally decreases as the number of known class increase (i.e. as openness decreases).This result is consistent with our hypothesis that smaller number of instances in each class can degrade performance. This phenomenon needs to be studied further with more datasets. The implications of this result is that knowing more classes is not enough but sufficient amount of data samples for each class is also needed.

Conclusion

In this paper we present an open world classification approach used for malware family classification. The approach



(a) Average area under ROC up to 10% FPR on MS-Challenge Dataset



(b) Average area under ROC up to 10% FPR on Android dataset

Figure 2: The relationship between the performance of openset classifier and number of known classes.

uses features extracted from the predicted class probabilities received from a classifier to train an outlier detector. The classifier and the outlier detector together form a system that is not only capable of distinguishing between known classes but is also capable of identifying instances arising from unknown (never before seen) classes. The evaluation results show that our approach compares favorably in accuracy with previous works on open world classification and multi-class outlier detection techniques. The evaluation also shows that our approach takes less time for both training and test.

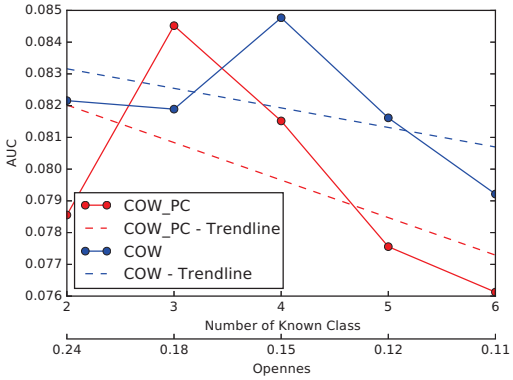


Figure 3: Experiment carried out on an down-sampled MS-Challenge dataset so as to have a comparable number of per class instances with the android dataset.

References

Bodesheim, P.; Freytag, A.; Rodner, E.; Kemmler, M.; and Denzler, J. 2013. Kernel null space methods for novelty detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3374–3381.

Bodesheim, P.; Freytag, A.; Rodner, E.; and Denzler, J. 2015. Local novelty detection in multi-class recognition problems. In *2015 IEEE Winter Conference on Applications of Computer Vision*, 813–820. IEEE.

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

Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; and Sander, J. 2000. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, 93–104. ACM.

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3):15.

Hassen, M., and Chan, P. K. 2017. Scalable function call graph-based malware classification. In *7th Conference on Data and Application Security and Privacy*, 239–248. ACM.

Jain, L. P.; Scheirer, W. J.; and Boult, T. E. 2014. Multi-class open set recognition using probability of inclusion. In *European Conference on Computer Vision*, 393–409. Springer.

2017. Knfst. https://github.com/cvjena/knfst.

2017. Libsvm-openset. https://github.com/ljain2/libsvm-openset.

Android malware genome project. http://www.malgenomeproject.org/.

2015. Microsoft malware classification challenge (big 2015). https://www.kaggle.com/c/malware-classification. [Online; accessed 27-April-2015].

Ramaswamy, S.; Rastogi, R.; and Shim, K. 2000. Efficient algorithms for mining outliers from large data sets. In *ACM SIGMOD Record*, volume 29, 427–438. ACM.

Rieck, K.; Trinius, P.; Willems, C.; and Holz, T. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19(4):639–668.

Scheirer, W. J.; de Rezende Rocha, A.; Sapkota, A.; and Boult, T. E. 2013. Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence* 35(7):1757–1772.

Tan, P.-N.; Steinbach, M.; and Kumar, V. 2006. *Introduction to data mining*. Pearson Education India.

Zhang, J. 2013. Advancements of outlier detection: A survey. *EAI Endorsed Trans. Scalable Information Systems* 1(1):e2.