COW: Malware Classification in an Open World

***Abstract*—A large number of new malware families are released on a daily basis. However, most of the existing works in the malware classification domain are still designed and evaluated under a closed world assumption in which the families of malware seen during testing are the same as the kind seen during training. In this paper we propose an open world malware classification algorithm. 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. We use a new feature space to help identify unknown malware families. Our proposed approach is more accurate and scales better on two evaluation datasets when compared to existing algorithms.**

***Keywords*-pen world classification, malware classification, open set recognitionpen world classification, malware classification, open set recognitiono**

I. 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. For instance, according to AV-Test Institute[2] more than 120 million new malware instances were registered in 2016.

When working in this domain it is, therefore, important to have classifiers that are not only capable of identifying if a malware sample belongs to a known malware family but able to determine if a sample does not belong to any of the unknown families and label it as $unknown$.

In this paper we present a classifier system that in addition to discriminating between the known classes is also capable of identifying instances that arise from unknown classes it did not see during training. 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.

The rest of this paper is organized in the following manner. We start with the discussion of related research works in Section II. Then we present our proposed approach in Section III. Finally, we discuss the results from experimental evaluation in Section IV.

II. 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.[16] in which malware samples are clustered into families. The authors use the distance of a test instance to cluster centroids to determine which class a malware sample belongs. If the distance to the closest cluster centroid is greater than some threshold, then that instance is deemed to be an outlier. The limitation of this approach comes from the use of unsupervised method for identifying between the known families which does not take advantage of label information that might be present for instances of the known classes.

Open world classification has aspects similar to anomaly detection problems. The main similarity is that in both cases there is an interest in identifying outlier samples. Anomaly detection is a well researched area and has found application in the security realm such as intrusion detection. For example, Ramaswamy et al. [15] propose a distance based approach which looks at the distance of an instance from it k^{th} nearest neighbor. Instances are ranked based on this distance and n furthest instances are determined to be anomalous.

An interesting observation pointed out by Breunig et al. [10] is that previous techniques such as the one discussed above take a global view of anomalies (i.e. the same threshold is use for identifying anomalies applies to all points.) However, some points that appear to be normal when considering a global view can be anomalous relative to their neighborhood. The density based approach proposed by [10] accounts for such anomalies. They first define the density of a point as inverse of the average distance to the k^{th} nearest points. Then the anomaly score of that point is the ratio of the density of the point to the average density of the k nearest points. Many more approaches have been proposed for this problem; we direct the reader to [11], [18], [19] for a more detailed survey of the different anomaly detection techniques.

However, there are two main differences between the open world classification and anomaly detection problems. The first difference arises from the learning task. In anomaly detection the task is only detecting anomalous samples and not in learning to discriminate the normal samples into different classes. The second difference comes from the assumption which is taken in anomaly detection that outlier instances are rare [18]. This assumption, however, does not hold in the case of the problem we are trying to solve as the number of instances from unknown classes is bound to be many.

A research that relates anomaly detection and open world classification is proposed for the image recognition domain by Scheirer et al. [17]. They propose a modification to one-class SVM, called 1-vs-set SVM, to handle an open world scenario for image recognition. Although their work is very close to the open world problem we are interested in addressing, it is not an exact solution. In our problem domain we are interested in having multi-class classifiers that in addition to classifying between the known classes are also capable of rejecting samples from unknown classes, which means that there are more than one known classes. In their case, however, they are solving the problem where their is one known class and the unknown instances can arise from several classes.

Multi-class anomaly detection called Kernel Null Foley-Sammon Transform (KNFST) proposed in [8] shares one of the objectives of the open world classification problem, which is to identify instances that do not belong to any of the known classes. It, however, does not determine to which of the known classes the instance belongs. In this approach all training instances that belong to the same class are projected to a single point in a new space, resulting in one target point per class. To determine if a test instance is an outlier the instance is first projected in to this new space and its distance to each class's target point is computed. The distance to the closest target point is then used as the anomaly score. Bodesheim et al. [7] 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.

A multi-class open set classifier framework PI_SVM is presented by Jain et al. [13]. They present an approach for fitting a single-class probability model over the known class scores from a discriminative binary classifier. A collection of binary classifiers are used, one for each known class. The resulting per-class probability inclusion model is used as an outlier measure.

III. Approach

Generally speaking 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. To verify this we conducted an experiment in which we trained a classifier on a set of known classes C^k and then tested it on both new instances from the known classes as well as instances from unknown classes C^u . Figure 1 shows a histogram of the maximum predicted class probability for instances from C^k and C^u. Where the maximum predicted probability is defined as:

$$P_{max} = \max_{c \in C^k} Pr(y_i = c \mid \vec{x_i}) \tag{1}$$

In case of instances from known classes, the figure on the left shows that the classifier makes prediction with max probability of 1.0 for more than half of the instances. In case of unknown class instances, the figure on the right shows that max probability is lower. This can be interpreted as the classifier generally displays less confidence when making predictions for instances from unknown classes as opposed to known classes.

A. New Derived Features

Based on these observations 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} defined in Equation 1. 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

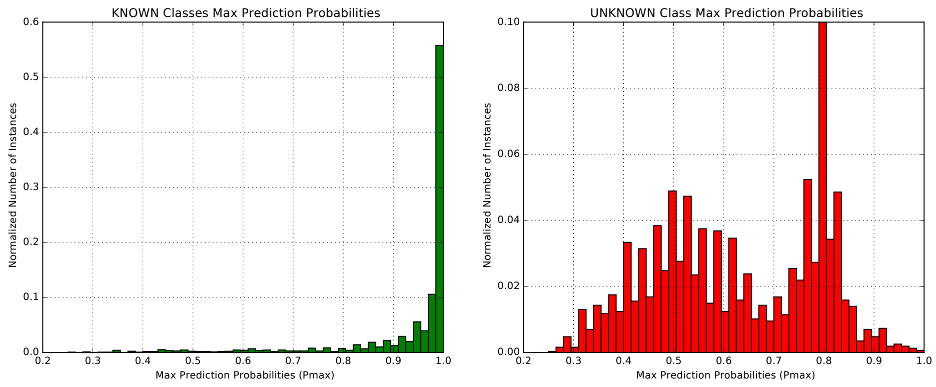


Figure 1: Histogram of maximum predicted class probabilities when a classifier classifies instances from known and unknown classes. In case of known class instances the classifier makes a prediction with higher confidence reflected in terms of high maximum predicted class probability. Whereas in the case of unknown class instances the predictions are made with a lower confidence. Note the two plots have different scale in the y-axis.

entropy of the predicted class probabilities. Entropy quantifies the diversity in the predicted class probabilities(in other words tells us how unevenly distributed the predicted class probabilities are.) The entropy for probability distribution p over $\mid C^k \mid$ classes is defined as:

$$entropy(p) = -\sum_{j}^{|C^k|} p_j \log p_j \tag{2}$$

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 $X_{inlier}$;
```

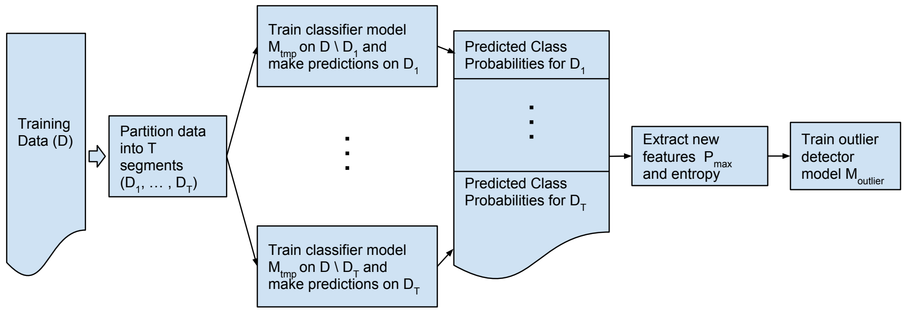
Algorithm 1 outlines the procedure used to extract new features from predicted class probabilities. The algorithm starts by splits the training set into T segments (line 1). For each segment, first a classifier is trained on data D excluding the instances in D_t (line 4). Then, predictions are made on 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.

B. 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 C^k from those belonging to classes from C^u. One of the challenges of trying to recognize instances from unknown classes comes from the fact that we can only get training data representing the known classes. 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 outlier detection method to train a model on the the predictions on 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 and Figure 2 outline 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. Once Algorithm 1 finishes it returns X_{inlier}, which is the training set represented in terms of the new features. 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).

Figure 3 illustrates how predictions using COW are made.



Training Data (D)
Partition data into T segments (D1, ... , DT)
Train classifier model Mtmp on D \ D1 and make predictions on D1
Train classifier model Mtmp on D \ DT and make predictions on DT
Predicted Class Probabilities for D1
Predicted Class Probabilities for DT
Extract new features Pmax and entropy
Train outlier detector model Moutlier

Figure 2: Training outlier detector model $M_{outlier}$ in COW based on Algorithm 2.

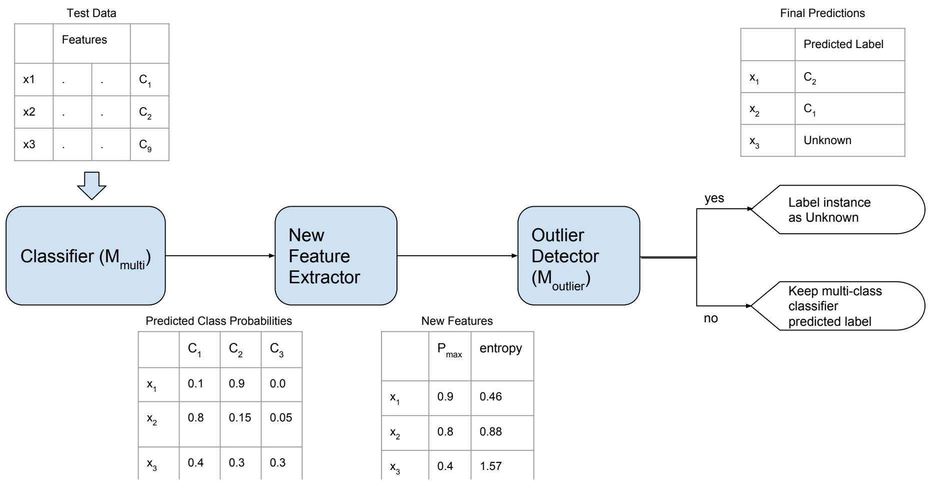


Figure 3: Making predictions using COW.

Algorithm 2: Training COW

```
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
    $M_{outlier}$: 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 $M_{outlier}$;
```

For example, let's assume our training set consisted of instances from classes c_1, c_2, and c_3. During testing we receive instances x_1,x_2, and x_3. In this example, x_1 and x_2 belong to known classes c_1 and c_2, respectively, while x_3 belongs to c_5 which was not present in our training data. First, M_{multi} is used to predict the class labels and class probabilities for each instance. Next, the new features, P_{max} and $entropy$, are extracted from the predicted class probabilities. This is then given as input to $M_{outlier}$ to make a prediction on whether an instance belongs to a known class or an unknown class. If an instance is predicted to be an outlier, then an $Unknown$ label will be assigned to it as in the case of x_3. Otherwise, the predicted class label by M_{multi} is going to be assigned to the instance as in the case of x_1 and x_2.

C. Per Class Classification in an Open World (COW_PC)

The reader might have noticed that 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 to predict 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, one for each class in C^k. By doing so we aim to address the aforementioned challenges.

The training procedure for COW_PC, Algorithm 3, is

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 $ListM_{outlier}$;
```

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 predictions about a test instance. In the case of COW_PC, if the instance is predicted as class $c \in C^k$ by M_{multi} then the outlier detector for class c ($M_{outlier_c}$) is then used to determine if the instance is indeed from class c or if it is from an unknown class.

IV. Experimental Evaluation

A. Evaluation Datasets

In this paper we used two datasets for evaluating the proposed approaches. The first is the Microsoft Malware Classification Challenge (MS-Challenge) dataset [4]. This dataset contains 10,867 disassembled windows malware binaries (about 197GB) from 9 malware families/classes. Our disassembled file parser were able to properly parse 10,260 of the samples. Table I shows the class distribution.

The second dataset used is from the Android Malware Genome Project [1]. This dataset contains different families of malicious android apps. The original dataset contained classes with very few samples. This presented a challenge when performing the evaluation because we setup the open world experiments. For instances, when a class is used as part of the known class set we add 75% of its instances to training set and the remaining 25% to the test set. Therefore, in the cases where a class that has smaller than 40 samples the number of instance in both training and test sets will be very small. For this reason we use only those families from Andriod dataset which have at least 40 samples. The class distribution is shown in Table II.

Table I: Microsoft malware dataset class distribution

Malware Family Name	Number of Samples
Ramnit	1513
Lollipop	2470
Kelihos ver3	2936
Vundo	446
Simda	34
Tracur	294
Kelihos˙ver1	387
Obfuscator.ACY	1168
Gatak	1012

Table II: Class distribution of the subset of Android Malware Genome project dataset used for evaluation.

Malware Family Name	Number of Samples
DroidKungFu3	309
AnserverBot	187
BaseBridge	122
DroidKungFu4	96
Geinimi	69
Pjapps	58
KMin	52
GoldDream	47
DroidDreamLight	46

B. Simulating Open World Scenario

In an open world scenario a classifier is trained on instances from a set of known classes C^k and tested on data that contains instances from both known classes C^k and unknown classes C^u (i.e. classes that the classifier was not trained on).

To simulate this 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 these classes in C^u to the test set. As for instances from classes in C^k, we randomly added 75% of them to training set and the remaining 25% to the test set.

C. Malware Features

The malware features we used to train our classifier models in these experiments are based on the research of Hassen and Chan [12]. 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. This representation consists of the vertex and edge frequencies. Even though the original paper uses a two-level classifier system with these features, we use a single classifier in our experiments.

D. Choice of Classifier and Outlier Detection Algorithms

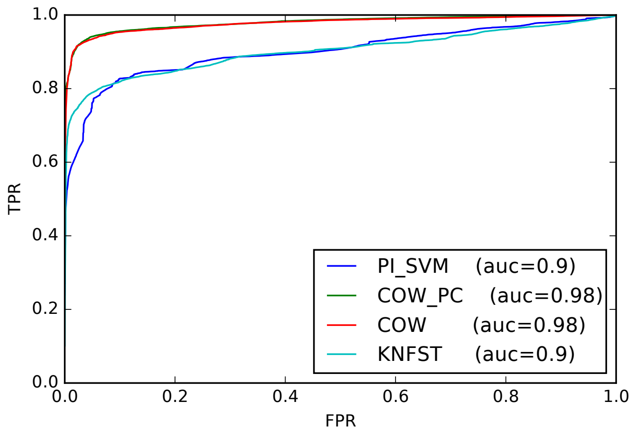
The design of our proposed approach allows for the use of any off the shelf classifiers and outliers detectors for the classifier and outlier detector component. In the following evaluation experiments we use Random Forest (RF) [9] for the multi-class classifier. Because of the relatively high dimensionality of our two datasets RF is a good choice for a classifier. RF tends to perform well when the set of features is large. Each splitting feature at each node in each tree of a random forest is selected from a random subset of features. This property of RF makes it usually suitable for high dimensional dataset. Another aspect of RF that made it suitable for our application is its fast training speed.

For the outlier detector we experimented with two different outlier detection algorithms: Isolation Forest [14] and an outlier detector we built by extending KD-Tree based Kernel Density Estimation algorithm in scikit-learn [3]. In our experiments we observed that the KD-Tree based Density Estimation performed slightly better than Isolation Forest. Hence, all of our results presented in the following sections use KD-Tree based Kernel Density Estimation as the outlier detector.

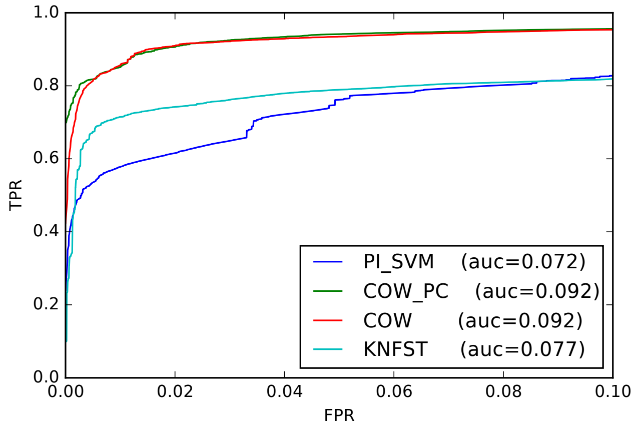
E. Accuracy Identifying Known Classes Instances from Unknown Class Instances

We start the evaluation of our work by comparing our approaches with two other previous approaches on open set recognition: PI_SVM [13], [6] and KNSFT [8], [5]. For both of the previous approaches we use the original authors implementation of the algorithms with a few wrapper methods of our own to make them work with our experiment framework. The authors implementation of KNSFT [5] only provides output on whether an instances belongs to C^k or C^u but does not provide prediction in which of the known classes an instance belongs. Consequently, in these experiments we report only result on determining whether an instance belongs to C^k or C^u.

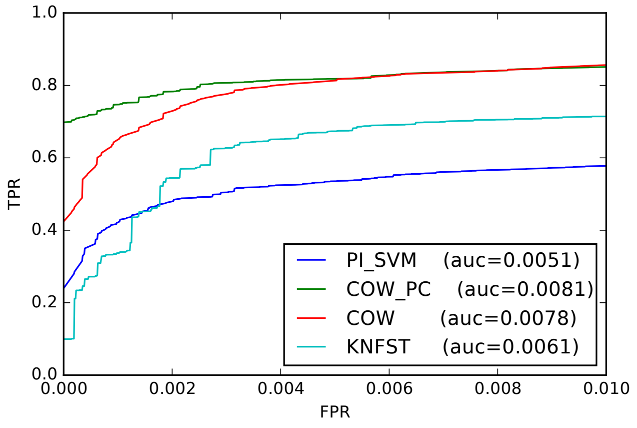
We performed 10 experiments for each approach. In each experiment the training and test sets are created as described in section IV-B by setting the number of unknown classes $|C^u|$ to be 3. The training set is used to train the models. We then use the learned model to generate an outlier score for 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 4 shows the result of these experiments carried out on the Microsoft Malware Challenge dataset in the form of the average ROC. Figure 5, on the other hand, shows



(a) ROC up to 100% FPR



(b) ROC up to 10% FPR



(c) ROC up to 1% FPR

Figure 4: Average ROC from 10 runs for distinguishing between instances from known and unknown classes on Microsoft Mawlare Challenge Dataset.

Table III: 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%

the result of similar experiments on the Android Malware Genome Project dataset.

When generating these ROC figures, we treat identifying the 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.

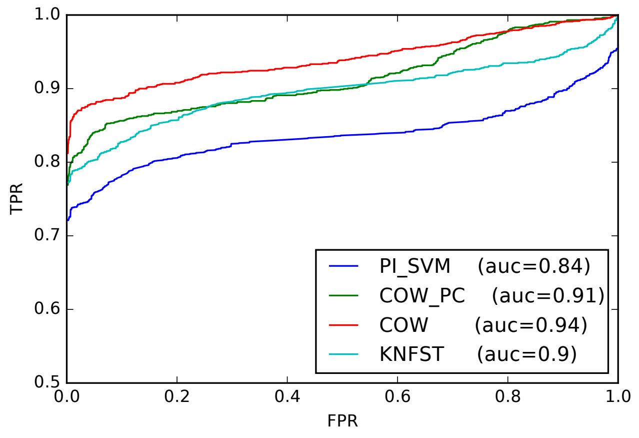
We would like to highlight certain observations from the results in Figures 4 and 5. First, our proposed approaches perform better compared to PI_SVM and KNFST. For example on the MS-Challenge dataset, at a 10% false positive rate (i.e. where 10% of instances from unknown classes get predicted as belonging 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. Similarly, our approaches record better AUC and TPR for 1% and 100% FPR.

The second observation we would like to highlight is that our approach achieves a relatively high TPR even at 0% FPR. Table III 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.

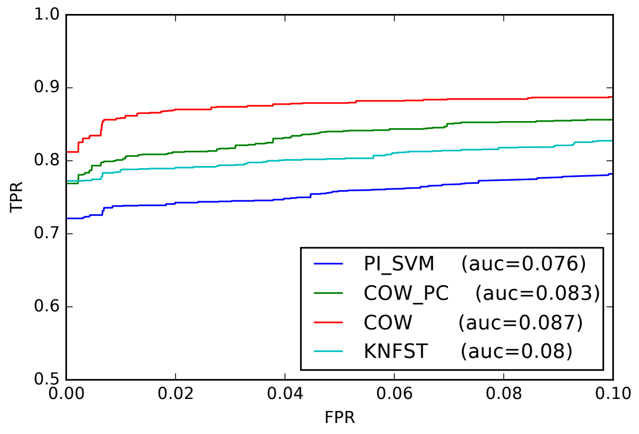
The final observation is that COW and COW_PC perform worse on the Android dataset than on the MS dataset. We assume this is due to the smaller number of instances per class on the Android dataset compared to the MS dataset. The discussion in Section IV-J explains this assumption in more detail.

F. Accuracy Discriminating Between Known Classes

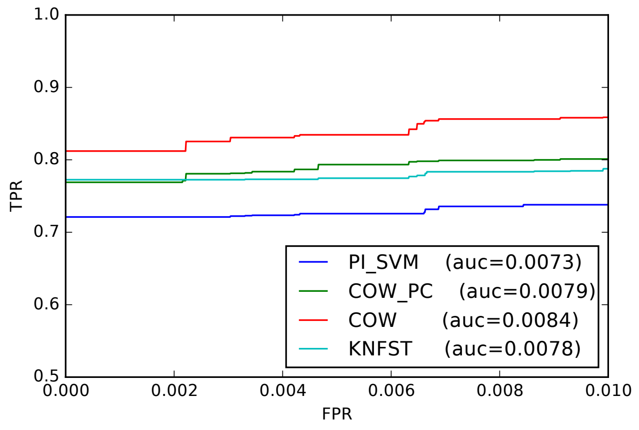
The results presented so far show how well the different approaches perform on the task of distinguishing between known class instances and unknown class instances. In addition to this, an open world classifier also needs to be able to discriminate between the known classes. In other



(a) Average ROC up to 100% FPR



(b) Average ROC up to 10% FPR



(c) Average ROC up to 1% FPR

Figure 5: Average ROC from 10 runs for distinguishing between instances from known and unknown classes on the Android Malware Genome Project.

words, the classifier needs to identify which of the known classes, if any, a test instance belongs to.

To evaluate this we run experiments where the open world dataset for each experiment is created in accordance to Section IV-B. 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 recored the precision, recall and f-score values of each class in the known class. We then calculate the final weighted average values of these metrics by using the fraction of the size of each known class in the test sets as weights. These results are reported in Table IV.

Since KNFST does not have the capability to discriminate between the known classes, we report results for COW, COW_PC, and PI_SVM. All three algorithms have one hyperparameter that specifies the threshold for discriminating between known and unknown class instances. We propose using validation set to perform binary search over the hyperparameter space using f-score as the search metrics. F-score is useful in this case because generally speaking f-score will have a global maxima. Hence as long as the this maxima is within the search space a value close to it can be found using binary search.

We observe in Table IV that for both datasets our approaches perform better in discriminating between the known classes. On the MS-Dataset COW and COW_PC record an F-Score of 0.91 and 0.90 compared to 0.85 of PI_SVM. A similar observation can be made for the Android dataset.

G. *Why is our approach performing better?*

In our opinion the main reason behind our approach's good performance comes from the fact that we transform the feature space for the outlier detection problem into a new feature space. This new feature space of our approach consists of the entropy and the maximum value of the predicted class probabilities (P_{max}), discussed in Section III-A.

Figure 6 shows a plot of a sample training data for the outlier detector in terms of the maximum predicted class probability and entropy. There are six subplots in the figure one for each known class in the training set. Each subplot shows a scatter plot of the instances that where correctly predicted as belonging to the class in a 2d space of P_{max} and $entropy$. In case of COW_PC each of these subplots represent the training instances given to the outlier detector of that class.

Because the instances in the scatter plots are tightly packed, the outlier detector has an easier job of learning tight boundaries to identify predictions that belong to a known class. This in turn should translate to a good performance in identifying known class instances from unknown class instances.

This is also a good place to remark on the performance comparison between COW and COW_PC. The main motivation behind COW_PC is to improve on COW by dedicating separate outlier detector models for each class. However, we observe mixed results so far. COW performs better in some instances and COW_PC in others. We hypothesize that one reason could be that the region learned by the single outlier detector of COW is overlapping with the regions learned but the outlier detectors of each class in COW_PC. For example, when we look at the plot for classes Vundo and Gatak in Figure 6 we see that regions that contain the instances overlap. Of course this is not the case for all classes, but if it for so for a considerable number of classes then the advantage of COW_PC might not become apparent.

H. *Efficiency Comparison with other Approaches*

Another important aspect to consider, apart from accuracy, when choosing a Machine Learning method is efficiency. To compare the time efficiency of our approaches with PI_SVM and KNFST, we record the training and test times when running the experiments in Section IV-E. Table V presents the average 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.

The main observation to make from Table V is that our two approaches, COW and COW_PC, seem to scale better compared to the other two algorithms. On the smaller android dataset, all four algorithms seem to have comparable training time and our approaches have a slightly faster test time. On the larger MS dataset, however, while our two approaches still scale well, both PI_SVM and KNFST record longer training and test times. Our two approaches record only a doubling in training and test time for an almost 10 fold increase in size of the training and test set from the Android dataset to the MS dataset.

I. *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 as 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 in the manner outlied in Section IV-B. 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

Table IV: Weighted average precision, recall and f-score of known classes.

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

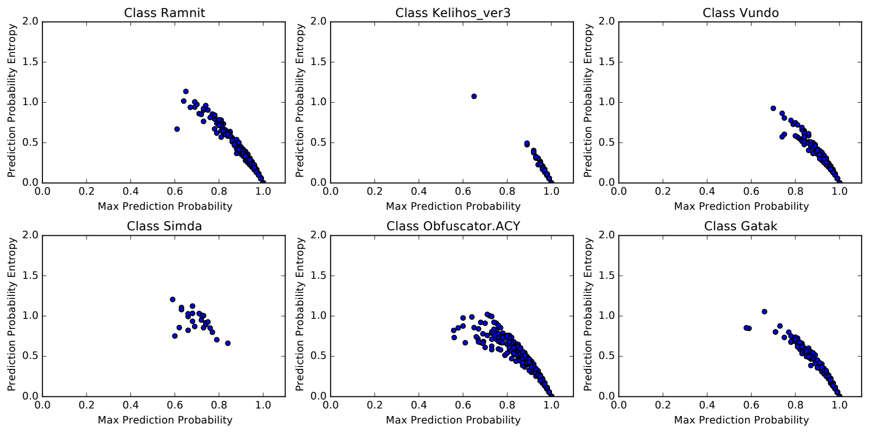


Figure 6: The new feature space used as input the the outlier detector on the MS Dataset.

Table V: 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

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 in [17] as:

$$openness = 1 - \sqrt{\frac{2 \times \mid training\ classes \mid}{\mid test\ classes \mid + \mid target\ classes \mid}} \tag{3}$$

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.

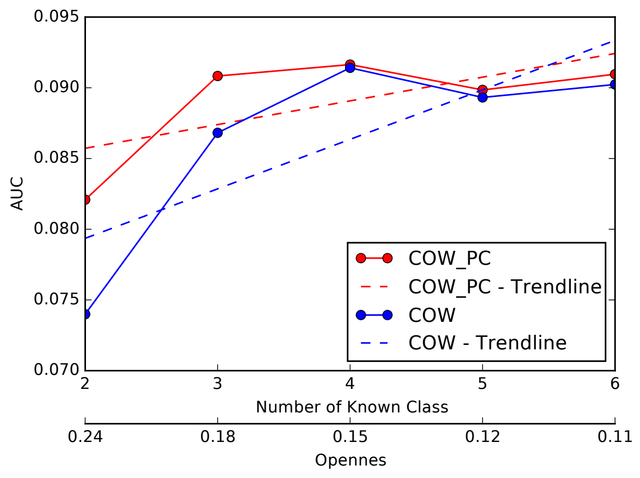
In the context of these experiments, increasing the number of known classes during training while keeping the number of unknown classes seen during testing constant will result in decreasing openness. We expect that this decrease in openness should in turn result in improved performance. This expectation aligns with what Jain et al. [13] show in their experiments.

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. The more surprising result comes in the case of the Android dataset, Figure 7b. In this case we see that performance actually degrades as openness decreases (i.e. as number of known classes increases). Next section attempts to explain the possible reasons behind this result.

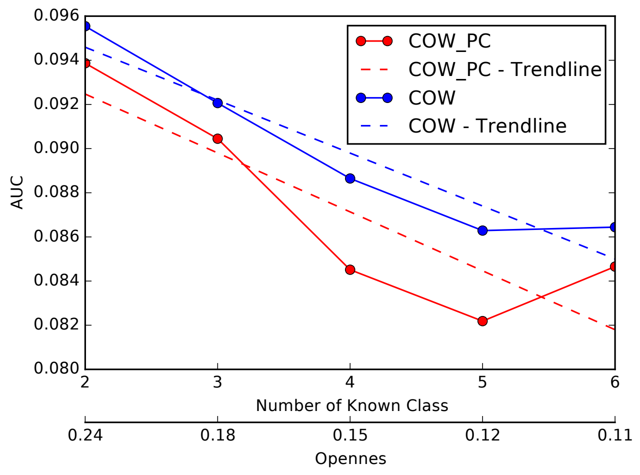
J. Relation between Class Size, Openness and AUC

In this section we present our hypothesis to explain the unexpected decrease in AUC when we decrease openness in the Android dataset in Section IV-I. This hypothesis also helps explain why our approach records lower AUC in case of the android dataset as presented in Section IV-E.

We hypothesis 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 (in Table I) with the Android dataset (in Table II), 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



(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 7: The relationship between the performance of openset classifier and number of known classes.

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 also results in decreased performance when comparing results on MS-Challenge dataset with Android dataset.

We carried out experiments to evaluate this hypothesis by down-sampling MS-Challenge dataset to have a comparable number of instances for each class with the Android dataset. The results in Figure 8 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.

V. Conclusion

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

We evaluate our work by simulating an open world scenario. 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.

The two new features discussed in Section III-A can be improved. As future work we plan to investigate more features that can be derived from the predicted class probabilities to improve the performance of COW and COW_PC.

References

[1] Android malware genome project. http://www.malgenomeproject.org/.

[2] Av-test malware statistics. http://www.av-test.org/en/statistics/malware/.

[3] scikit-learn. http://scikit-learn.org/stable/.

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

[5] Knfst. https://github.com/cvjena/knfst, 2017.

[6] Libsvm-openset. https://github.com/ljain2/libsvm-openset, 2017.

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

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

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

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

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

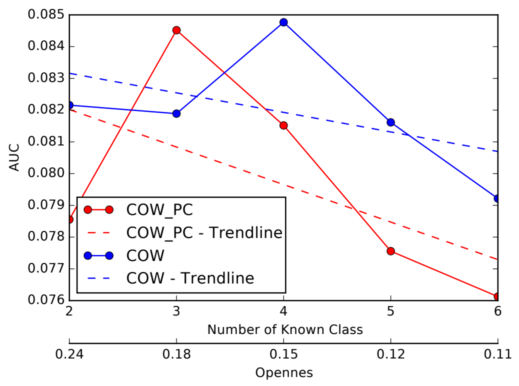


Figure 8: Experiment carried out on an down-sampled MS-Challenge dataset so as to have a comparable number of per class instances with the android dataset.

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

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

[14] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 413–422. IEEE, 2008.

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

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

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

[18] P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2006.

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