Expert Systems With Applications 212 (2023) 118590

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Fast & Furious: On the modelling of malware detection as an evolving data stream

Fabrício Ceschin [a,*], Marcus Botacin [a], Heitor Murilo Gomes [b], Felipe Pinagé [a], Luiz S. Oliveira [a], André Grégio [a]

[a] Federal University of Paraná – Cel. Francisco Heráclito dos Santos, 100 – Curitiba, Paraná, Brazil
[b] Victoria University of Wellington – Cotton Building, Victoria University of Wellington, Kelburn, Wellington 6012, New Zealand

ARTICLE INFO

Keywords:
Machine learning
Data streams
Concept drift
Malware detection
Android

ABSTRACT

Malware is a major threat to computer systems and imposes many challenges to cyber security. Targeted threats, such as ransomware, cause millions of dollars in losses every year. The constant increase of malware infections has been motivating popular antiviruses (AVs) to develop dedicated detection strategies, which include meticulously crafted machine learning (ML) pipelines. However, malware developers unceasingly change their samples' features to bypass detection. This constant evolution of malware samples causes changes to the data distribution (i.e., concept drifts) that directly affect ML model detection rates, something not considered in the majority of the literature work. In this work, we evaluate the impact of concept drift on malware classifiers for two Android datasets: DREBIN (≈130K apps) and a subset of AndroZoo (≈285K apps). We used these datasets to train an Adaptive Random Forest (ARF) classifier, as well as a Stochastic Gradient Descent (SGD) classifier. We also ordered all datasets samples using their VirusTotal submission timestamp and then extracted features from their textual attributes using two algorithms (Word2Vec and TF-IDF). Then, we conducted experiments comparing both feature extractors, classifiers, as well as four drift detectors (Drift Detection Method, Early Drift Detection Method, ADaptive WINdowing, and Kolmogorov–Smirnov WINdowing) to determine the best approach for real environments. Finally, we compare some possible approaches to mitigate concept drift and propose a novel data stream pipeline that updates both the classifier and the feature extractor. To do so, we conducted a longitudinal evaluation by (i) classifying malware samples collected over nine years (2009–2018), (ii) reviewing concept drift detection algorithms to attest its pervasiveness, (iii) comparing distinct ML approaches to mitigate the issue, and (iv) proposing an ML data stream pipeline that outperformed literature approaches, achieving an improvement of 22.05 percentage points of F1Score in the DREBIN dataset, and 8.77 in the AndroZoo dataset.

1. Introduction

Countering malware is a major concern for most networked systems, since they can cause billions of dollars in loss (SecurityVentures, 2018). The growth of malware infections (CTONetworks, 2017) enables the development of multiple detection strategies, including machine learning (ML) classifiers tailored for malware, which have been adopted by the most popular AntiViruses (AVs) (Gandotra et al., 2014; Jordaney et al., 2017; Kantchelian et al., 2013). However, malware samples are very dynamic pieces of code – usually distributed over the Web (Chang et al., 2013) and constantly evolving to survive – quickly turning AVs into outdated mechanisms that present lower detection rates over time. This phenomenon is known as concept drift (Gama et al., 2014), and requires AVs to periodically update their ML classifiers (Ceschin et al., 2018). Malware concept drift is an understudied problem in the literature, with the few works that address it usually focusing on achieving high accuracy rates for a temporally localized dataset, instead of aiming for long-term detection due to malware evolution. Moreover, the community considers ML a powerful ally for malware detection, given its ability to respond faster to new threats (Gibert et al., 2020).

Being the most used operating system worldwide, Android has more than 2 billion monthly active devices (Fergus Halliday, 2018), with almost 40% of prevalence in the operating system market, surpassing

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

[*] Corresponding author.
E-mail addresses: fjoceschin@inf.ufpr.br (F. Ceschin), mfbotacin@inf.ufpr.br (M. Botacin), heitor.gomes@vuw.ac.nz (H.M. Gomes), fapinage@inf.ufpr.br (F. Pinagé), lesoliveira@inf.ufpr.br (L.S. Oliveira), gregio@inf.ufpr.br (A. Grégio).

https://doi.org/10.1016/j.eswa.2022.118590
Received 9 February 2021; Received in revised form 13 August 2022; Accepted 13 August 2022
Available online 22 August 2022

Microsoft Windows in 2018 (StatCounter, 2018). As a widespread platform, Android is more affected by malware evolution and distribution, rendering its AVs vulnerable to concept drift effects, which makes the need to adapt existing solutions.

In this work, we evaluate the impact of concept drift on malware classifiers for two Android datasets: DREBIN (Arp et al., 2014) (≈130 K apps) and a subset of AndroZoo (Allix et al., 2016) (≈350 K apps). For our longitudinal evaluation, we collected malware samples over nine years (2009–2018) and used them to train an Adaptive Random Forest (ARF) classifier (Gomes et al., 2017), as well as a Stochastic Gradient Descent (SGD) classifier (Pedregosa et al., 2011). Our goal is to answer the following questions: (i) is concept drift a generalized phenomenon and not an issue of a particular dataset? (ii) is it important to update the feature extractor (and not only the classifier) when a concept drift is detected? (iii) how can we consider the feature extractor in the malware detection pipeline? (iv) which drift detector is the best for this task? (v) is concept drift somehow related to changes in the Android ecosystem? To do so, we ordered all datasets samples using their VirusTotal (VirusTotal, 2018) submission timestamp and then extracted features from their textual attributes using two algorithms (Word2Vec (Mikolov et al., 2013) and TF-IDF (Salton et al., 1975)). After that, we conducted experiments comparing both feature extractors, classifiers, as well as four drift detectors (Drift Detection Method (Gama et al., 2014), Early Drift Detection Method (Baena-Garćıa et al., 2006), ADaptive WINdowing (Bifet & Gavaldà, 2007), and Kolmogorov–Smirnov WINdowing (Raab et al., 2020)) to determine the best approach for real environments. We also compare some possible approaches to mitigate concept drift and propose a novel method based on the data stream pipeline that outperformed current solutions by updating both the classifier and the feature extractor. We highlight the need for also updating the feature extractor, given that it is as essential as the classifier itself to achieve increased detection rates due to new features appearing over time. Therefore, we propose a realistic data stream learning pipeline, including the feature extractor in the loop. We also show with our experiments that concept drift is a generalized phenomenon in Android malware. Finally, we discuss the impact of changes on the Android ecosystem in our classifiers by comparing feature changes detected over time, and the implications of our findings. It is worth notice that all code and datasets used is publicly available,[1] as well as the implementation of our data stream learning pipeline using `scikit-multiflow` (Montiel et al., 2018),[2] an open-source ML framework for stream data.

This article is organized as follows: we compare our work with the literature in Section 2; we introduce our methodology in Section 3; we describe the machine learning background in Section 4; we present the experiments results in Section 5; we discuss our findings in Section 6, and draw our conclusions in Section 7.

2. Related work

The literature on malware detection using machine learning is extensive (Gandotra et al., 2014). Usually, the primary concern of most works is to achieve 100% of accuracy using different representations and models, ignoring the fact that malware samples evolve as time goes by. Few papers consider concept drift in this context, such as Masud et al. which were (at the best of our knowledge) the first to treat malware detection as a data stream classification problem like us, but proposing an ensemble of classifiers trained from consecutive chunks of data using v-fold partitioning of them and reducing classification error compared to other ensembles (Masud et al., 2008). They also presented a feature extraction and selection technique for data streams that do not have any fixed feature set (the opposite of our approach) based on information gain. Bach et al. combined two models: a stable one, based on all data, and a reactive one, based on a short window of recent data, to determine when to replace the current stable model by computing the difference in their accuracy, assuming that the stable one performs worse than the reactive when the concept changes (Bach & Maloof, 2008), very similar to what we do when the drift detector reaches a warning level.

Singh et al. proposed two measures to track concept drift in static features of malware families (which we do by using concept drift detectors, but in a malware detection task): relative temporal similarity (based on the similarity score between two time-ordered pairs of samples and are used to infer the direction of concept drift) and meta-features (based on summarization of information from a large number of features) (Singh et al., 2012), claiming to provide paths to further exploration of drift in malware detection models. Narayanan et al. presented an online machine learning based framework, named DroidOL to handle concept drift and detect malware (Narayanan et al., 2016) using control-flow sub-graph features in an online classifier, which adapts to the malware drift by updating the model more aggressively when the error is large and less aggressively, otherwise, which can also be done by drift detectors. Deo et al. proposed the use of Venn-Abers predictors to measure the quality of classification tasks and identify concept drift (Deo et al., 2016).

Jordaney et al. presented Transcend, a framework to identify concept drift in classification models which compares the samples used to train the models with those seen during deployment, computing algorithm credibility and confidence to measure the quality of the produced results and detect concept drift (Jordaney et al., 2017). Anderson et al. have shown that, by using reinforcement learning to generate adversarial samples, it is possible to retrain a model and make these attacks less effective, also protecting it against possible drifts, given that this technique hardens a model against worst-case inputs (Anderson et al., 2018), something that can be improved even more by retraining the feature extractor as we do. Pendlebury et al. reported that some results are inflated by spatial bias, caused by the wrong distribution of training and testing sets, and temporal bias, caused by incorrect time splits into these same sets (Pendlebury et al., 2018), which is one of the reasons why we collected the timestamps in our datasets and used data streams. They introduced an evaluation framework called Tesseract to compare malware classifiers in a realistic setting and confirmed that earlier published results are biased. We reinforce that in our findings by comparing different experiments with a real-world simulation using data streams. Ceschin et al. compared a set of experiments that use batch machine-learning models with ones that take into account the change of concept (data streams), emphasizing the need to update the decision model immediately after a concept drift occurs (Ceschin et al., 2018). In contrast, we do not update only the decision model, but also the feature extractor when drift occurs. The authors also show that the malware concept drift is related to their concept evolution due to the appearance of new malware families.

Mariconti et al. created MaMaDroid, an Android malware detection system that uses a behavioral model, in the form of a Markov chain, to extract features from the API calls performed by an app (Onwuzurike et al., 2019). The solution proposed was tested with a dataset containing 44 K samples, collected over six years, and presented a good detection rate and the ability to keep its performance for long periods (at least five years according to their experiments). We increased the length of the observation window by using malware samples collected over nine years. Cai et al. compared their approach (Cai, 2018; Cai & Jenkins, 2018), which uses 52 selected metrics concerning sensitive data accesses via APIs (using dynamic analysis) as features, with MaMaDroid. According to their experiments, their approach managed to remain stable for five years, while MaMaDroid only kept the same performance for two years. A very similar approach was compared with

[1] https://www.kaggle.com/datasets/fabriciojoc/fast-furious-malware-data-stream.

[2] https://github.com/fabriciojoc/scikit-multiflow.

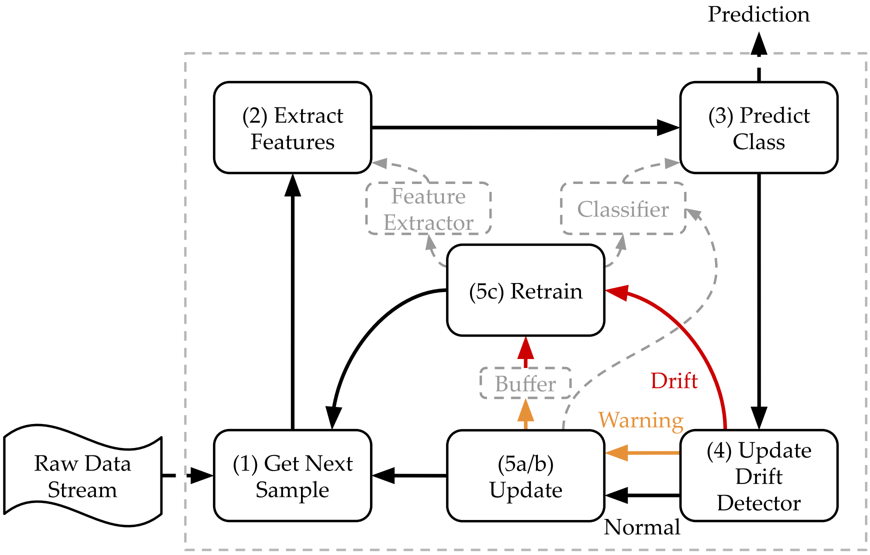


Fig. 1. Data Stream Pipeline. Every time a new sample is obtained from the data stream, its features are extracted and presented to a classifier, generating a prediction, which is used by a drift detector that defines the next step: update the classifier or retrain both classifier and feature extractor.

other four methods in literature and all of them presented an overall f1score below 72% (Fu & Cai, 2019).

Xu et al. proposed DroidEvolver, an Android malware detection system that can be automatically updated without any human involvement, requiring neither retraining nor true labels to update itself (Xu et al., 2019). The authors use online learning techniques with evolving feature sets and pseudo labels, keeping a pool of different detection models and calculating a juvenilization indicator that determines when to update its feature set and each detection model. We compared our approach with DroidEvolver and showed that it outperformed it. Finally, Gibert et al. presented research challenges of state-of-the-art techniques for malware classification, exemplifying the concept drift as one of them Gibert et al. (2020).

Zhang et al. designed APIGraph, a framework to detect evolved Android malware that groups similar API calls into clusters based on the Android API official documentation (Zhang et al., 2020). Applying the aforementioned technique to other Android malware classifiers, the authors reduced the labeling efforts when combined with Tesseract (Pendlebury et al., 2018).

Finally, different from other approaches listed, Massimo Ficco presented an ensemble detector that exploits diversity in the detection algorithms by using both generic and specialized detectors (trained to detect certain malware types). The author also presents a mechanism that explores how the length of the observation time window can affect the detection accuracy and speed of different combinations of detectors during the detection (Ficco, 2022).

Our main contribution in this work is to apply data stream based machine learning algorithms to malware classification, proposing an important improvement in the pipeline that makes feature vectors correspond to the actual concept. The proposed solution, to the best of our knowledge, was not considered before and is as important as updating the classifier itself. To test and validate our proposal, we use two datasets containing almost 480 K android apps, showing that it outperforms traditional data stream solutions. We also include an analysis of how certain features change over time, correlating it with a cybersecurity background.

3. Methodology

3.1. Threat model and assumptions

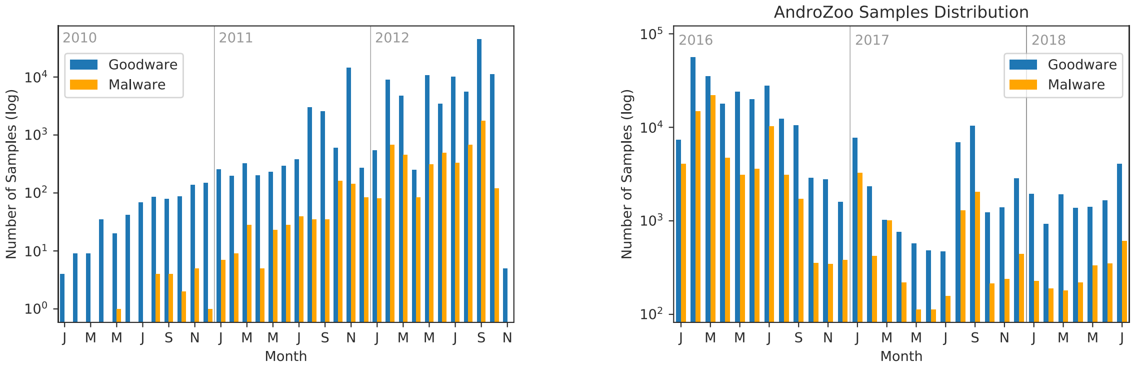
Our threat model considers an antivirus engine (AV) for the Android platform since it is the market share leader. Thus, successful malware infections affect a large number of users. For scientific purposes, we considered an AV entirely based on Machine Learning (ML) as implemented by many of the malware detectors cited in Section 2. In practice, however, it does not imply that an AV must use only this detection method: it can be complemented with any other detection approach envisioned by the AV vendor. Our detection model is completely static (features are retrieved directly from the APK files) because static malware detection is the most popular and fastest way to triage malware samples. Similarly to the above discussion, the use of static detectors do not imply that an AV should not use dynamic components, but that our research focuses on improving the static detection component. In our proposed AV model, the APK files are inspected as soon as they are placed in the Android filesystem, i.e., it does not rely on any other system information except the APK files themselves. It is worth emphasizing that our goal is not to implement an actual AV but to highlight the need for updating ML models based on classifiers. Therefore, we simulated the behavior of an online AV using offline experiments that use data streams, simplifying our solution implementation. The details of the simulated ML model are presented below.

3.2. Data stream

Since our goal is to evaluate the occurrence of concept drift in malware classifiers, we analyzed malware detection evolution over time using a data stream abstraction for the input data. In a traditional data stream learning problem that includes concept drift, the classifier is updated with new samples when a change occurs—usually the ones that caused the drift (Gama et al., 2014). Our data stream pipeline also considers the feature extractor under changes, according to the following five steps shown in Fig. 1:

1. Obtain a new sample X from the raw data stream;
2. Extract features from X using a feature extractor E, trained with previous data;
3. Predict the class of the new sample X using the classifier C, trained with previous data;
4. With the prediction from C, update the drift detector D to check the drift level (defined by authors (Montiel et al., 2018));
5. According to the drift level, three paths can be followed, all of them making the pipeline restarts at **Step 1**:

   a **Normal:** incrementally **update** C with X;



(a) **DREBIN dataset distribution by month.** Very few samples are shown in 2009 (not shown), 2010, 2013 (not shown) and 2014 (not shown). The majority of them are in 2011 and 2012.

(b) **AndroZoo dataset distribution by month.** The majority of samples are in 2016, with fewer malware showing in 2017. February 2016 and March 2016 are the months with more prevalence of goodware and malware, repectively.

Fig. 2. Datasets distribution over time.

b **Warning:** incrementally **update** C with X and add X to a buffer;

c **Drift: retrain** both E and C using only the data collected during the warning level (from the buffer build during this level), creating a new extractor and classifier.

3.3. Datasets

A challenge to detect concept drift in malware classifiers is to properly identify the sample's date to allow temporal evaluations. Since malware samples are collected in the wild and they may be spreading for some time, no actual creation date is available. As an approximation for it, we considered each sample's first appearance in VirusTotal (VirusTotal, 2018), a website that analyzes thousands of samples every day. Malware samples were ordered by their "first seen" dates, which allows us to create a data stream representing a real-world scenario, where new samples are released daily, thus requiring malware classifiers to be updated (Pendlebury et al., 2018).

In our experiments, we considered attributes vectors provided by the authors of DREBIN (Arp et al., 2014), composed of ten textual attributes (API calls, permissions, URLs, etc.), which are publicly available to download and contain 123,453 benign and 5560 malicious Android applications. We show DREBIN's distribution in Fig. 2(a). We also considered a subset of Android applications reports provided by AndroZoo API (Allix et al., 2016), composed of eight textual attributes (resources names, source code classes and methods, manifest permissions etc.) and contains 267,342 benign and 80,102 malicious applications. We show the distribution of our AndroZoo subset in Fig. 2(b): it keeps the same goodware and malware distribution as the original dataset, which originally is composed by most of 10 million apps.

It is important to notice that we are using the attributes that were already extracted statically from them Arp et al. (2014) and Allix et al. (2016), since we do not have access to applications binaries, packages, or source codes. In addition, the timing information provided by the authors of the DREBIN (Arp et al., 2014) and VirusTotal differs. According to Arp et al. (2014), their samples were collected from August 2010 to October 2012. However, our version is based on VirusTotal appearance date, which showed us that very few samples were already analyzed by VirusTotal (48 in 2009) and some of them were just analyzed after the establishment of the collection (37 in 2013 and 2014), probably following the dataset publicly release. We do not show these samples in Fig. 2 for better visualization.

Furthermore, both datasets reflect two characteristics of the real world that challenge the development of efficient ML malware detectors: (i) long-term variations (2009–2014, for DREBIN and 2016–2018 for AndroZoo); and (ii) class imbalance. For example, in DREBIN, whereas more than 40 K goodware were collected in Sep/2009, only 1750 malware samples were collected in the same period. Whereas class imbalance is out of this work's scope, we considered both datasets as suitable for our evaluations due to the long-term characteristic, which challenges ML classifiers to learn multiple concepts over time. Finally, to evidence the difference of both datasets, we created heat maps containing the prevalence of a subset of malware families created using the intersection of families from them (54 families) (see Fig. 3).

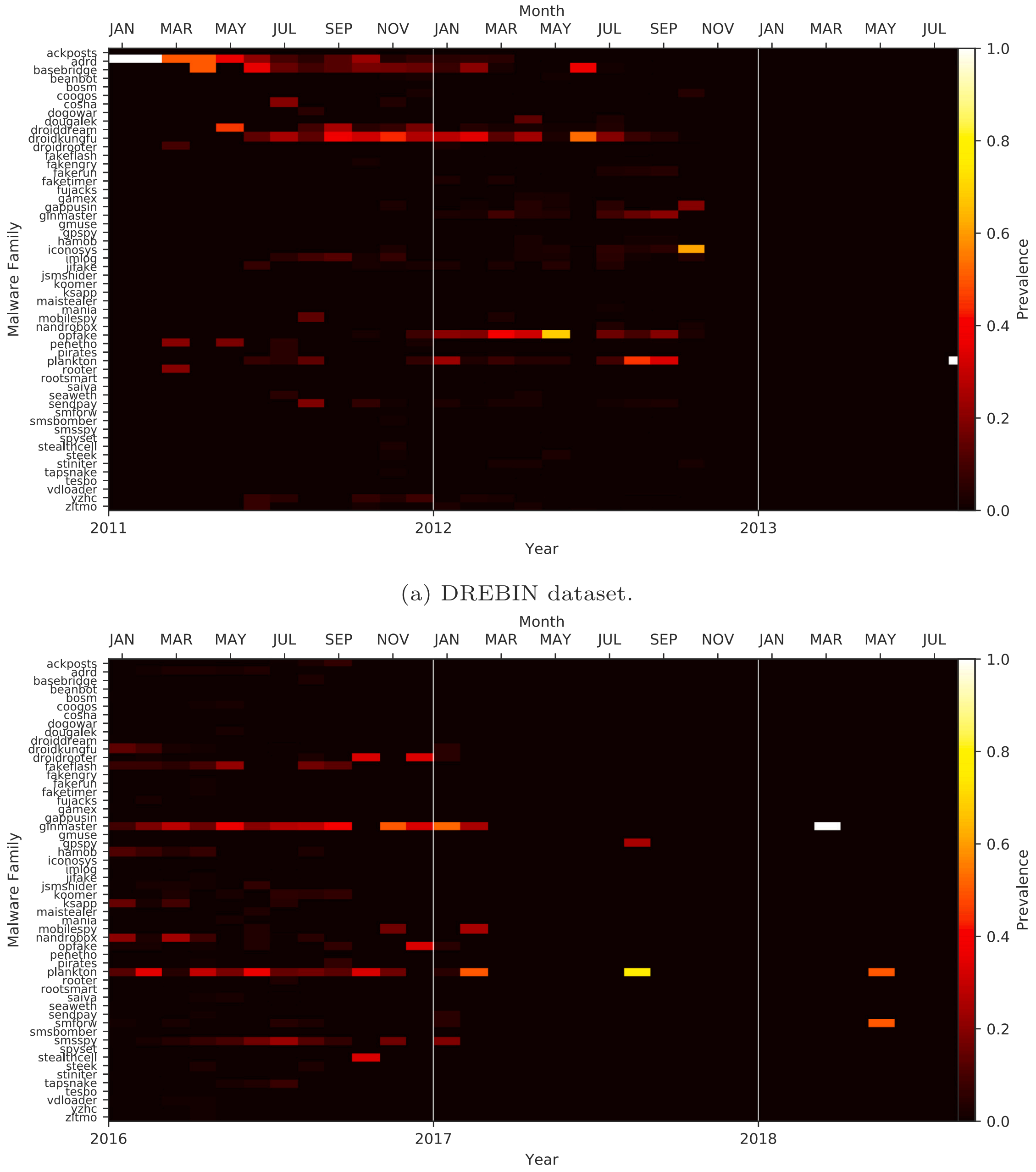
4. Machine learning algorithms

4.1. Representation

A typical way to represent malware for ML is to create a vector using the sample textual attributes, such as API calls, permissions, URLs, providers, intents activities, service receivers, and others. In our work, we represented them using Word2Vec (Mikolov et al., 2013) and TF-IDF (Salton et al., 1975), since they are widely used representations for text classification. Besides, both use the training data to generate the representation of all samples, which allows us to test our hypothesis that the drift affects not only the classifier but also the feature extractor.

4.2. Concept drift detectors

We used four concept drift detection algorithms in our work: DDM (Drift Detection Method (Gama et al., 2014)), EDDM (Early Drift Detection Method (Baena-Garcı́a et al., 2006)), ADWIN (ADaptive WINdowing (Bifet & Gavaldà, 2007)), and KSWIN (Kolmogorov–Smirnov WINdowing) (Raab et al., 2020). Both DDM and EDDM are online supervised methods based on sequential error monitoring, i.e., each incoming example is processed separately estimating the sequential error rate. Therefore, they assume that the increase of consecutive error rate suggests the occurrence of concept drift. DDM directly uses the error rate, while EDDM uses the distance-error rate, which measures the number of examples between two errors (Baena-Garcı́a et al., 2006).



(a) DREBIN dataset.

(b) AndroZoo dataset.

Fig. 3. Malware family distribution. Intersection of families from both datasets shows evidences of class evolution, given that they are very different.

These errors trigger two levels: warning and drift. The warning level suggests that the concept starts to drift, then an alternative classifier is updated using the samples which rely on this level. The drift level suggests that the concept drift occurred, and the alternative classifier build during the warning level replaces the current classifier. ADWIN keeps statistics from sliding windows of variable size, used to compute the average of the change observed by cutting them in different points. If the difference between two windows is greater than a pre-defined threshold, a concept drift is detected, and the data from the first window is discarded (Bifet & Gavaldà, 2007). KSWIN uses a sliding window of fixed size to compare the most recent samples of the window with the remaining ones by using Kolmogorov–Smirnov (KS) statistical test (Raab et al., 2020). Different from the other two methods, ADWIN and KSWIN has no warning level, which made it necessary to adapt our data stream ML pipeline. For instance, when a change occurs in ADWIN, out of the window data is discarded and the remaining samples are used to retrain the both the classifier and feature extractor. Finally, when a change occurs in KSWIN, only the most recent samples of the window are kept and used to retrain the classifier and feature extractor.

4.3. Classifiers

In all evaluations, we developed classification models leveraging Word2Vec and TF-IDF representations and normalized using a MinMax technique (Pedregosa et al., 2011). For all cases, we used as classifiers Adaptive Random Forest (Gomes et al., 2017) without its internal drift detectors (working as a Random Forest for data streams), since its widespread use in the malware detection literature and has the best overall performance (Ceschin et al., 2018), and Stochastic Gradient Descent (SGD) classifier (Pedregosa et al., 2011), which is one of the

Table 1
Cross-validation. Mixing past and future threats is the best scenario for AVs in both datasets.

Classifier	Algorithm	DREBIN dataset				AndroZoo dataset			
		Accuracy	F1Score	Recall	Precision	Accuracy	F1Score	Recall	Precision
Random Forest	Word2Vec	99.09%	88.73%	82.12%	**96.48%**	90.52%	76.27%	66.03%	90.29%
	TF-IDF	**99.23%**	**90.63%**	**85.85%**	96.31%	**91.54%**	**79.30%**	**70.25%**	**91.03%**
SGD	Word2Vec	98.29%	78.28%	70.90%	87.36%	85.41%	60.05%	47.52%	81.54%
	TF-IDF	**98.63%**	**83.26%**	**78.49%**	**88.66%**	**88.74%**	**71.57%**	**61.43%**	**85.73%**

fastest online classifiers in scikit-learn (Pedregosa et al., 2011). Both the classifier and drift detectors were configured using the same hyperparameters proposed by the authors (Gomes et al., 2017; Montiel et al., 2018).

5. Experiments

5.1. The best-case scenario for AVs (ML cross-validation)

In the first experiment, we classify all samples together to compare which feature extraction algorithm is the best and report baseline results. We tested several parameters for both algorithms and fixed the vocabulary size in 100 for TF-IDF (top-100 features ordered by term frequency) and created projections with 100 dimensions for Word2Vec, resulting in 1000 and 800 features for each app in both cases, for DREBIN and AndroZoo, respectively. All results are reported after 10-fold cross-validation procedures, a method commonly used in ML to evaluate models because its results are less prone to biases (note that we are training new classifiers and feature extractors at every iteration of the cross-validation process). In practice, folding the dataset implies that the AV company has a mixed view of both past and future threats, despite temporal effects, which is the best scenario for AV operation and ML evaluation.

Table 1 presents the results obtained in this experiment for both DREBIN and AndroZoo datasets using Adaptive Random Forest (ARF) and Stochastic Gradient Descent (SGD), highlighting the performance of TF-IDF, which was better than Word2Vec in all metrics, except in precision when classifying the DREBIN dataset. It means that Word2Vec is slightly better to detect goodware (particularly in DREBIN dataset) since its precision is higher (i.e., fewer FPs) and TF-IDF is better to detect malware due to its higher recall (i.e., less FNs). In general, we conclude that TF-IDF is better than Word2Vec since its accuracy and f1score are higher. Moreover, we notice that its f1score is not as high as the accuracy, which is near 100%, indicating that one of the classes (malware) is more difficult to predict than the other (goodware). Regardless of small differences, we observe that ML classifiers perform significantly well when samples of all periods are mixed, even in a more complex dataset such as AndroZoo, since they can learn features from all periods.

5.2. On classification failure (temporal classification)

Although currently used classification methodology helps reducing dataset biases, it would demand knowledge about future threats to work properly. AV companies train their classifiers using data from past samples and leverage them to predict future threats, expecting to present the same characteristics as past ones. However, malware samples are very dynamic, thus this strategy is the worst-case scenario for AV companies. To demonstrate the effects of predicting future threats based on past data, we split our datasets in two: we used the first half (oldest samples) to train our classifiers, which were then used to predict the newest samples from the second half. The results of Table 2 indicate a drop in all metrics when compared to the 10-fold experiment in both DREBIN and AndroZoo datasets and also suggest the occurrence of concept drift on malware samples, given that the recall is much smaller than the previous experiment.

Due to dataset imbalance in both datasets, we notice a bias toward goodware detection (very small accuracy decrease) and significant qualitative differences for f1score and recall (much worse than before). The precision score was very similar to the cross-validation experiment, and better in the case of TF-IDF when classifying DREBIN, showing that goodware samples remain similar in the "future" while malware samples evolved. Word2Vec and TF-IDF lost, in average, about 16 percentage points of their f1score in DREBIN and about 19 percentage points for this same metric in AndroZoo. In addition, the model's recall rates significantly dropped in both Word2Vec and TF-IDF when classifying DREBIN and AndroZoo, regardless of the models used. This indicates that detecting unforeseen malware only from a single set of past data is a hard task. These results highlight the need of developing better continuous learning approaches for effective malware detection.

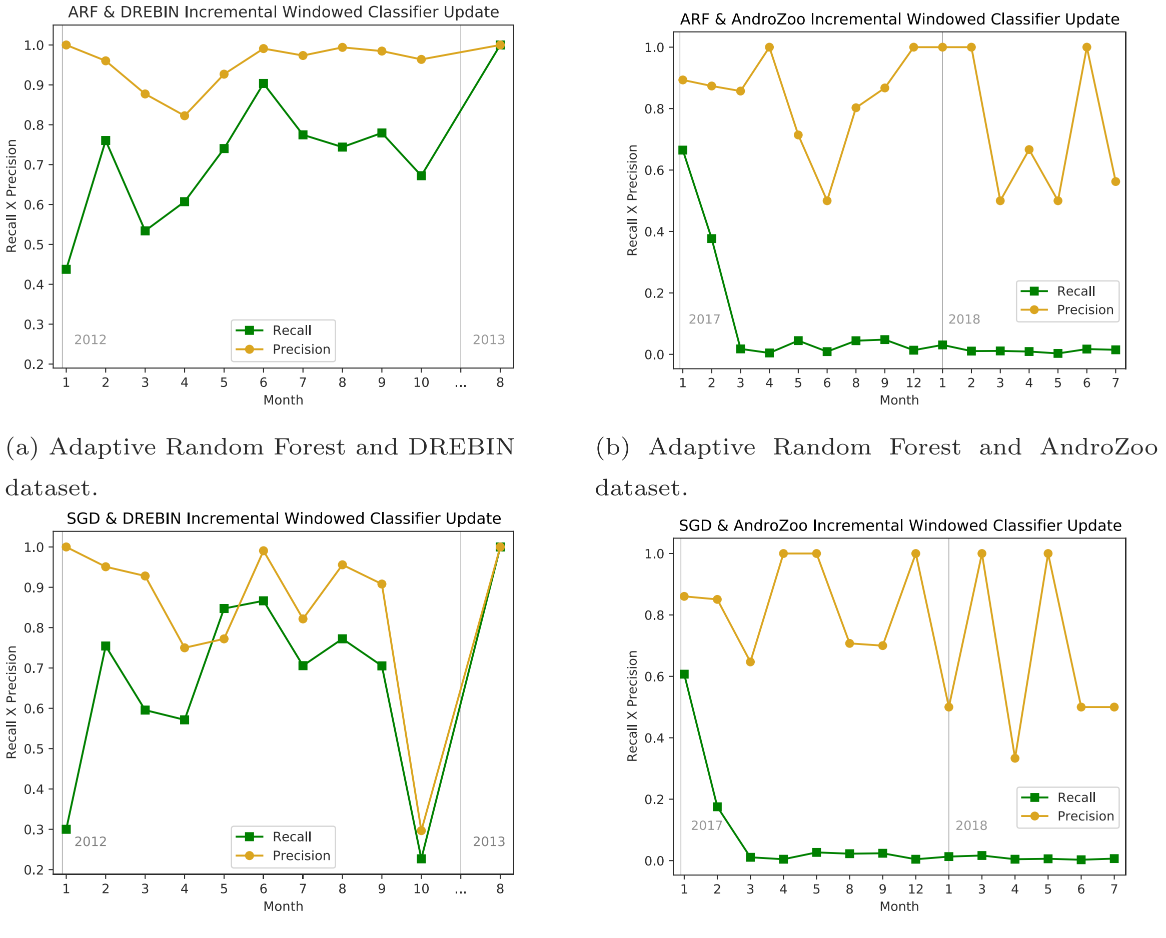
5.3. Real-world scenario (windowed classifier)

Since static classifiers are a bad strategy, AV companies adopt continuous updating procedures as samples are collected and identified. From a ML perspective, they adopt an incremental stream learning method (Pinage et al., 2016), which we call Incremental Windowed Classifier (IWC). Notice that this is the same approach used by other authors in the literature, but they propose the use of distinct attributes and features only, instead of a new stream pipeline (Cai, 2020; Zhang et al., 2020). To evaluate the impact of this approach, we divided the datasets into two groups, one containing samples released until a certain month of a year for training, and the other with samples released one month after that said month for testing. For example, considering Jan/2012, the training set contained samples that were created until Jan/2012 (cumulative) and the validation set contained samples created only in Feb/2012 (a month later). We tested both Adaptive Random Forest (ARF) (Gomes et al., 2017) and Stochastic Gradient Descent (SGD) classifier (Pedregosa et al., 2011) with TF-IDF (100 features for each textual attribute), given its better performance in previous experiments, and excluded months with no samples. Every month, both classifier and feature extractor were retrained, generating new classifiers and feature extractors.

The results for the DREBIN dataset shown in Figs. 4(a) and 4(c) indicate a drop in both precision and recall rates. For Adaptive Random Forest, precision drops to about 85% in April/2012 and increases in the following months to almost 100% in August/2013. The Recall starts with the worst result (about 40%), in January/2012 and reaches almost 100% in August/2013. The average recall was 70.3% and average precision, 95.36%. For SGD, the worst precision is reported in October/2012 (about 30%), increasing to almost 100% in August/2013. The same happen with the recall, which drops to almost 20% in October/2012 and increases to almost 100% in August/2013. Finally, the average recall was 66.78% and average precision, 85.22%. On the one hand, the growth of precision and recall in some periods indicate that continuously updating the classifier might increase classification performance in comparison to using a single classifier tested with samples from a past period because the classifier learns new patterns for the existing concepts (features) that may have been introduced over time. On the other hand, the drop in the precision and recall rates in some periods indicate the presence of new concepts (e.g., new malware features) that were not fully handled by the classifiers. When looking at the AndroZoo dataset results, as shown in Figs. 4(b) and 4(d), it is

Table 2
Temporal evaluation. Predicting future threats based on data from the past is the worst-case for AVs.

Classifier	Algorithm	DREBIN dataset				AndroZoo dataset			
		Accuracy	F1Score	Recall	Precision	Accuracy	F1Score	Recall	Precision
Random Forest	Word2Vec	97.66%	62.58%	46.31%	96.47%	87.55%	53.95%	38.71%	88.96%
	TF-IDF	**98.20%**	**73.26%**	**58.36%**	**98.39%**	**88.20%**	**57.13%**	**41.71%**	**90.63%**
SGD	Word2Vec	97.52%	65.14%	55.08%	79.70%	85.81%	47.04%	33.44%	79.28%
	TF-IDF	**98.15%**	**75.18%**	**66.42%**	**86.61%**	**86.96%**	**52.79%**	**38.68%**	**83.07%**



(a) Adaptive Random Forest and DREBIN dataset.

(b) Adaptive Random Forest and AndroZoo dataset.

(c) SGD and DREBIN dataset.

(d) SGD and AndroZoo dataset.

Fig. 4. Continuous Learning. Recall and precision while incrementally retraining both classifier (Adaptive Random Forest and Stochastic Gradient Descent classifier) and feature extractor.

possible to note a drastic fall in recall as time goes by in both cases, dropping from almost 70% (ARF) and 60% (SGD) in January/2017 to less than 5% in March/2019, not exceeding 10% by the end of the stream. This indicates that AndroZoo is a much more complex dataset with very different malware in distinct periods, given that the recall did not behave as this same experiment using the DREBIN dataset. In contrast, the precision remained unstable. For Adaptive Random Forest, it reaches 100% in some periods, such as May/2017, and drops to almost 50%, in June/2017, with quite similar behavior from the DREBIN dataset. For SGD, it also reaches 100% in some periods, such as April/2017, and drops to almost 30% in April/2018. These results indicate that more than continuous retraining, AV companies need to develop classifiers fully able to learn new concepts.

5.4. Concept drift detection using data stream pipeline (Fast & Furious – F&F)

Although the previous results indicate that continuously learning from the past is the best strategy for detecting forthcoming threats, concept drift remains challenging, even more when considering results from the AndroZoo dataset. Therefore, we present an evaluation of a drift detector approach that could be deployed by AV companies to automatically update their classifier models. In our experiments, we used both Adaptive Random Forest (Gomes et al., 2017) and Stochastic Gradient Descent as the classifiers, DDM, EDDM, ADWIN, and KSWIN as drift detectors (Montiel et al., 2018) (with the same parameters as the authors), and TF-IDF (using 100 features for each textual attribute) as a feature extractor. In the DREBIN dataset, we initialized the base classifier with data from the first year, given that in the first months we have just a few malware showing up. In the AndroZoo dataset, we initialized it with data from the first month. We tested all drift detection algorithms in two circumstances when creating a new classifier, according to our data stream pipeline: (i) **Update:** we just update the classifier with new samples (collected in warning level or ADWIN window), which reflects the fastest approach for AV companies to react to new threats; (ii) **Retrain:** we extract all features from the raw data (also collected in warning level or ADWIN window) and all models are built from scratch again, i.e., both classifier and feature extractor are retrained and a new vocabulary is built based on the words in the training set for each textual attribute (more complete approach, but time-consuming for AV companies). To compare our solution with another similar in the literature, we implemented our version of DroidEvolver (Xu et al., 2019), replicating their approach using the same feature representation as ours. It is important to report here that we tried to use the authors' source code, but they were not working properly due to dependencies that could not be installed. Thus, we implemented an approximation of their method by analyzing their paper and code, using $\tau_0 = 0.3$ and $\tau_1 = 0.7$ with three compatible online learning methods from scikit-learn (Pedregosa et al., 2011): `SGDClassifier`, `PassiveAggressiveClassifier`, and `Perceptron`. Finally, during the execution of DroidEvolver, we noticed that it was detecting at least one drift in one of its classifiers each iteration, making it necessary to create a new parameter that checks for drift in an interval of *steps* (iterations), in our case 500 steps was selected according to our analysis.

Table 3 presents the results for DroidEvolver and our methods for both datasets. For DREBIN, when using Adaptive Random Forest, EDDM outperforms DDM methods' classification performance in all scenarios, which was the opposite when using SGD classifier. For both classifiers, ADWIN outperforms EDDM, DDM, and KSWIN, providing the best overall performance for retraining as well as for learning new features. However, only when using Adaptive Random Forest, ADWIN with the update is better for precision, making it slightly better in reducing false positives, and KSWIN is better for recall, making it better in reducing false negatives. Moreover, retraining both the feature extractor and classifier makes it detect fewer drift points than the updating approach (18 vs. 20 when using ARF, 13 vs. 14 when using SGD). Overall, SGD outperformed Adaptive Random Forest when classifying DREBIN, presenting an improvement in f1score of almost 16 percentage points. In comparison to DroidEvolver, our approach outperformed it in all metrics, with a relatively large margin. In Fig. 5, it is possible to observe the prequential error of Adaptive Random Forest with ADWIN using Update (Fig. 5(a)) and Retrain (Fig. 5(b)) strategies. Despite the Retrain strategy is similar to the update approach in some points, it has less drift points, a lower prequential error and, consequently, a better classification performance.

When classifying AndroZoo, DDM outperforms EDDM, which was not a good drift detector for this specific dataset. ADWIN with retraining was the best method again, detecting 50 and 33 drift points versus 78 and 30 when using the update approach with Adaptive Random Forest and SGD classifier, respectively, outperforming all other methods, including DroidEvolver again. Finally, these results suggest that AV companies should keep investigating specific samples to increase overall detection.

5.5. Multiple time spans

In the previous experiment, we showed that the use of drift detectors with the Retrain strategy improved the classification performance. Although the experiment simulates a real-world stream, there might be potential biases from the use of the very same training and test sets. By proposing a new experiment, we mitigated the risk of biasing evaluations, since we have tested our solutions under different circumstances by using multiple time spans and reporting the average result.

To evaluate the effectiveness of our approach in multiple conditions, we applied our data stream pipeline to different training and test sets of our dataset: we split the two datasets (sorted by the samples' timestamps) into eleven folds (thus 10 consecutive epochs), with every fold containing the same amount of data, similar to a k-fold cross-validation scheme. However, instead of using a single set for training as done in each iteration of k-fold cross-validation, we increment the training set i with the fold $i+1$, and remove it from the test set at every iteration. This way, we create a cumulative training set and simulate the same scenario as the previous experiment, but starting the stream in different parts. In the end, we produced ten distinct results that present the effectiveness of our method under varied conditions, which worked as a k-fold cross-validation. However, we focused on collecting the F1Score of each iteration. In the current experiment, we used all the methods presented in the previous section: both classifiers (Adaptive Random Forest and SGD), all drift detectors (DDM, EDDM, ADWIN, and KSWIN), and both methods ((U)pdate and (R)etrain). To accomplish a better presentation of the results, we chose a boxplot visualization containing the distribution of all F1Scores (black dots) and their average for each method (white dots), as shown in Fig. 6.

In Figs. 6(a) and 6(c), we present the results for the DREBIN dataset using Adaptive Random Forest and SGD as classifiers, respectively. The IWC method performed much better than the others with Random Forest; in the case of using ten folds to predict the last one, it reaches a better performance than the other methods, which indicates that the last fold may not be affected by concept drift. However, when we look at the other methods, KSWIN with retrain performed best, achieving a higher average F1Score and a low standard deviation. Also, as we saw in the previous section, SGD performed better than the Adaptive Random Forest applied to this dataset.

In Figs. 6(b) and 6(d), we present the results for the AndroZoo dataset using Adaptive Random Forest and SGD as classifiers, respectively. In this scenario, IWC performance is much worse than any other method, which we believe is evidenced due to the complexity of this dataset (almost 350 K samples). Again, in both classifiers, KSWIN with retrain method performed best, achieving the higher average F1Score, despite Adaptive Random Forest presenting better overall results and being almost 4 percentage points higher than SGD.

Finally, the analysis of the results allows us to observe that (i) the more data we have in the data stream, the most difficult it becomes for IWC to keep a good performance; (ii) using KSWIN drift detector with retrain is the most recommended method for static android malware detection data streams; and (iii) we ensure that our results do not have potential biases.

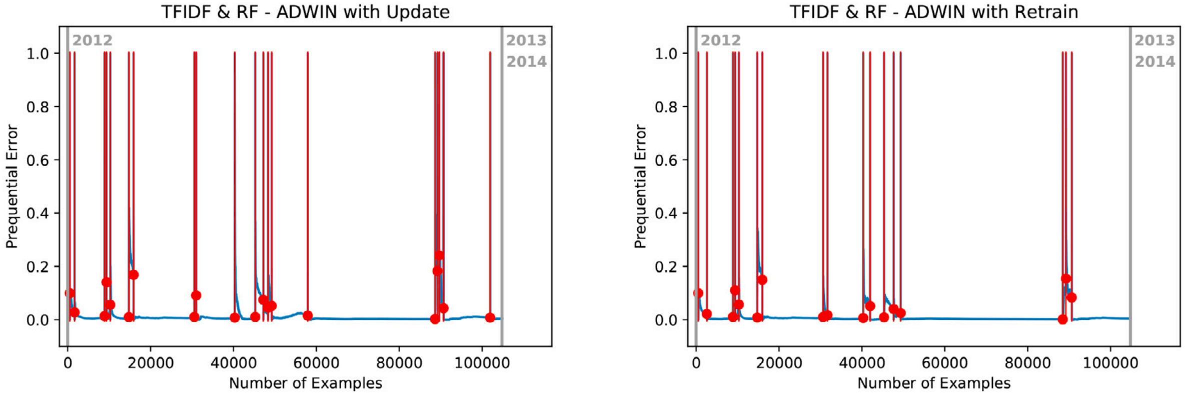
5.6. Understanding malware evolution

After we confirmed that concept drift is prevalent in these malware datasets, we delved into evolution details to understand the reasons behind it and the lessons that mining a malware dataset might teach us. To do so, we analyzed vocabulary changes over time for both datasets and correlated our findings of them.

DREBIN Dataset Evolution. Fig. 7(a) shows API calls' vocabulary change for the first six detected drift points that actually presented changes (green and red words mean introduced and removed from the vocabulary, respectively). We identified periodic trends and system evolution as the two main reasons for the constant change of malware samples. The first occurs because malware creators compromise systems according to the available infection vectors (e.g., periodic events exploitable via social engineering). Also, attackers usually shift their operations to distinct targets when their infection strategies become so popular that AVs detect them. Therefore, some features periodically enter and leave the vocabulary (e.g., `setcontentview`). The second occurs due to changes in the Android platform, causing reactions from malware creators to handle the new scenario, either by supporting newly introduced APIs (e.g., `getauthtoken` (Android, 2018a)), or by unsupporting deprecated and modified APIs (e.g., deleted `keyguard` (Android, 2018b) or the deprecated `fbreader` (FBReader, 2018) intent). Handling platform evolution is required to keep malware samples working on newer systems, whereas ensuring maliciousness. In this sense, the removal of a feature like `DELETE_PACKAGE` permission from the vocabulary can be explained by the fact that Android changed permission's transparency behavior to an explicit confirmation request (Android, 2016), which makes malware abusing those

Table 3
Overall results. Considering IWC, DroidEvolver (Xu et al., 2019), F&F (U)pdate and (R)etrain strategies with multiple drift detectors and classifiers (Random Forest and SGD).

Classifier	Method	DREBIN dataset					AndroZoo dataset				
		Accuracy	F1Score	Recall	Precision	Drifts	Accuracy	F1Score	Recall	Precision	Drifts
Model Pool	DroidEvolver (Xu et al., 2019)	97.27%	67.14%	59.14%	77.64%	69	87.09%	66.28%	56.17%	80.83%	22
Adaptive Random Forest	IWC	96.8%	80%	70.3%	95.36%	N/A	82.99%	11.4%	8.25%	79.61%	N/A
	DDM (U)	98.3%	79.19%	68.38%	94.04%	8	88.19%	70.84%	63.5%	80.11%	14
	DDM (R)	98.4%	80.54%	70.27%	94.32%	9	87.96%	69.92%	61.94%	80.27%	24
	EDDM (U)	98.53%	82.27%	71.84%	96.26%	27	78.28%	39.52%	37.23%	42.11%	118
	EDDM (R)	98.57%	82.85%	73.09%	95.6%	14	77.91%	39.31%	37.52%	41.28%	17
	ADWIN (U)	98.58%	82.66%	71.35%	**98.21%**	20	86.41%	65.86%	58.02%	76.15%	78
	ADWIN (R)	**98.71%**	**84.44%**	74.17%	98.02%	18	89.6%	**75.05%**	**69.23%**	81.93%	50
	KSWIN (U)	98.38%	80.82%	72.19%	91.80%	10	89.22%	71.78%	60.66%	87.88%	70
	KSWIN (R)	98.55%	83.01%	**74.96%**	93.00%	9	**89.66%**	73.22%	62.56%	**88.26%**	50
SGD	IWC	96.33%	73.40%	66.78%	85.22%	N/A	82.63%	9.16%	6.61%	75.71%	N/A
	DDM (U)	98.84%	87.56%	86.29%	88.88%	3	88.05%	71.17%	65.30%	78.20%	7
	DDM (R)	98.85%	87.67%	86.69%	88.66%	3	89.10%	73.81%	67.98%	80.73%	4
	EDDM (U)	98.85%	87.63%	86.43%	88.87%	11	82.99%	42.50%	32.97%	59.75%	85
	EDDM (R)	98.78%	87.05%	86.61%	87.49%	20	82.77%	41.80%	32.45%	58.72%	73
	ADWIN (U)	98.95%	88.68%	87.03%	90.38%	14	89.39%	73.98%	66.74%	82.97%	30
	ADWIN (R)	**99.00%**	**89.19%**	**87.38%**	**91.08%**	13	**89.49%**	**74.31%**	67.25%	**83.01%**	33
	KSWIN (U)	98.93%	88.45%	86.83%	90.13%	1	88.23%	72.41%	**68.34%**	77.00%	48
	KSWIN (R)	98.85%	87.71%	87.28%	88.15%	2	85.34%	67.05%	65.99%	68.14%	70



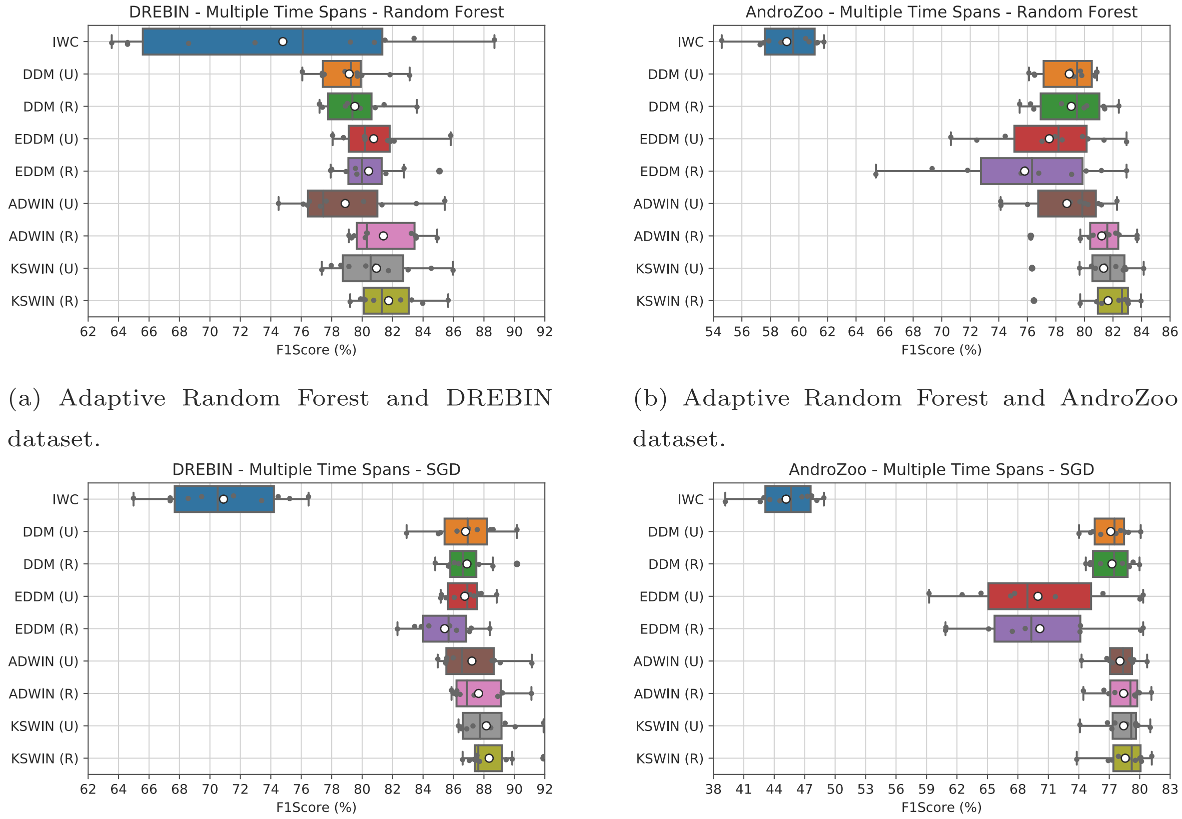
(a) Adaptive Random Forest with ADWIN and Update.

(b) Adaptive Random Forest with ADWIN and Retrain.

Fig. 5. F&F (U)pdate and (R)etrain prequential error and drift points as time goes by when using Adaptive Random Forest. Despite being similar in some points, fewer drift points are detected when retraining the feature extractor, reducing the prequential error and increasing classification performance.

features less effective. The most noticeable case of malware evolution involves SMS sending, a feature known to be abused by plenty of samples (Hamandi et al., 2012; Luo et al., 2013; Sarma et al., 2012). We identified that APIs were periodically added and removed from the vocabulary (e.g., sendmultiparttextmessage, part of the Android SMS subsystem (Android, 2018c), had its use restricted by Android over time, until being finally blocked (Cimpanu, 2018)). This will certainly cause modifications in newer malware and probably incur in classifiers' drifting once again. Therefore, considering concept drift is essential to define any realistic threat model and malware detector.

AndroZoo Dataset Evolution. We conducted the same experiments performed on the Drebin dataset on the AndroZoo dataset to evaluate dataset influence on concept drift occurrence, as shown in Fig. 7(b). We highlight that developing a malware classifier for the AndroZoo dataset is a distinct challenge than developing a malware classifier for the Drebin dataset because the AndroZoo dataset presents more malware families (1.166 distinct families according to Euphony (Hurier et al., 2017) labels) than the Drebin dataset (178 distinct families according to Euphony labels). The difference in the dataset complexity is reflected in the number of drift points identified for each experiment. Whereas we identified 18 drift points for the Drebin dataset, the AndroZoo dataset presented 50 drift points. Despite the difference in the number of drifting points, the reasons behind such drifts have been revealed very similar when we delve into vocabulary change details. When we look to the manifest_action vocabulary, we notice that the sms_received feature leaves and returns to the vocabulary many times. This behavior reflects the arms race between Google and attackers for SMS sending permission (Cimpanu, 2018). This same occurrence has been observed in the DREBIN dataset. Similarly, when we look to the manifest_category vocabulary, we notice that the facebook feature also leaves and returns to the vocabulary many times. This happens because the attackers often rely on Facebook name for distributing "Trojanized" applications that resemble that original Facebook app. Although Google often removes these apps from the official store, attackers periodically come up with new ways of bypassing AppStore's verification routines, thus resulting in the presented infection waves regarding the Facebook name. Finally, this same behavior is observed in the resource_entry vocabulary regarding the amazonaws feature. Attackers often store their malicious payloads on popular cloud servers to avoid detection (Rossow et al., 2013). Even though Amazon sinkhole malicious domains when detected, attackers periodically find new ways to survive security scans, thus also resulting in the observed vocabulary waves.



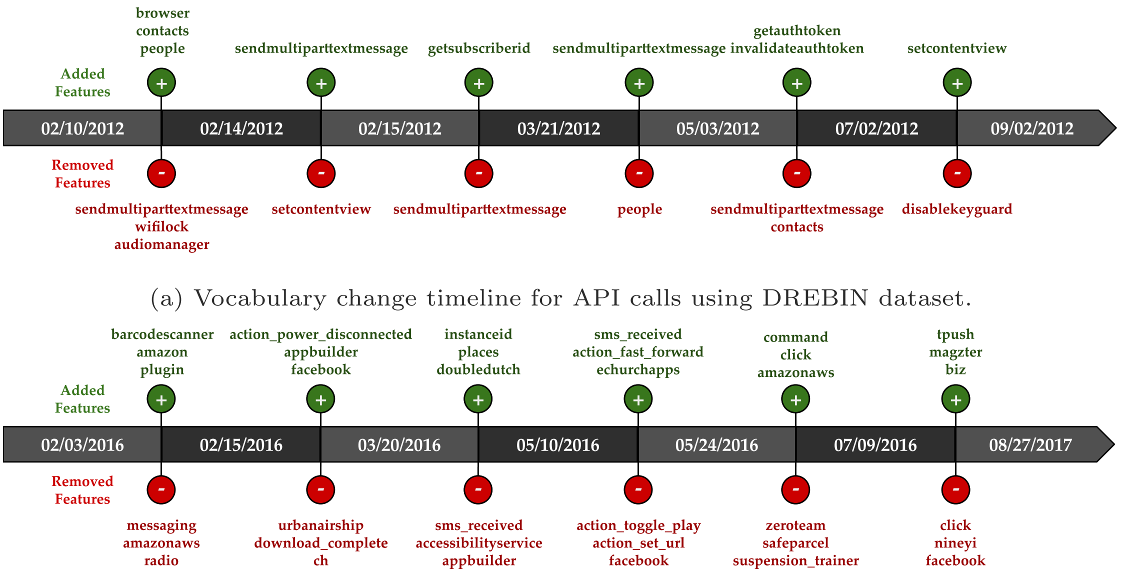
(a) Adaptive Random Forest and DREBIN dataset.

(b) Adaptive Random Forest and AndroZoo dataset.

(c) SGD and DREBIN dataset.

(d) SGD and AndroZoo dataset.

Fig. 6. Multiple Time Spans. F1Score distribution when experimenting with ten different sets of training and test samples. Black dots represent the distribution of all F1Scores, whereas white dots represent their average for each applied method.



(a) Vocabulary change timeline for API calls using DREBIN dataset.

(b) Vocabulary change timeline for manifest and resources using AndroZoo dataset.

Fig. 7. Vocabulary changes for both datasets. Many significant features are removed and added as time goes by.

The results of our experiments show that despite presenting distinct complexities, both datasets have undergone drift occurrences. This shows that concept drift is not a dataset-exclusive issue, but a characteristic inherently associated with the malware classification problem. Therefore, overall malware classification research work (and primarily the ones leveraging these two popular datasets) should consider concept drift occurrence in their evaluations to gather more accurate samples information.

6. Discussion

We here discuss our findings and their implications.

What the metrics say. While the accuracy of distinct classifiers is very similar in some experiments, we highlight that malware detectors must be evaluated using the correct metrics, as any binary classification. Therefore, we rely on f1score, recall, and precision to gain further knowledge on malware detection. AV companies should adopt the same reasoning when evaluating their classifiers. Experiments 5.2 and 5.3 indicate Android malware evolution, which imposes difficulties for established classifiers and corroborates the fact that samples changed. However, goodware generally kept the same concept over time, suggesting that creating benign profiles may be better than modeling everchanging malicious activities.

Feature drift, concept drift and evolution. Using a fixed ML model is not enough to predict future threats. In experiment 5.2, we notice a significant drop in f1score and recall (a metric that indicates the ability to correctly classify malware). Hence, a fixed ML model is prone to misclassify novel threats. Besides, continuous learning helps increasing detection but is still subject to drift, as shown by AndroZoo in experiment 5.3, when recall remains below 10% for a long period. If we compare experiments 5.3 and 5.4 when classifying DREBIN, it is possible to observe that the Incremental Windowed Classifier update still outperforms DDM with the update when using Adaptive Random Forest, but is not better than DDM with retraining (except in precision). However, when comparing it with EDDM and ADWIN, we notice that both are significantly better, and it is still highly prone to drift (Fig. 4). In response to these results, concept drift detectors help to improve classification performance. When comparing DDM with retraining and EDDM and ADWIN (with both update and retrain) to IWC (Table 3), we can see that the former outperformed the latter, advocating the need of using drift detectors. This is evident when using SGD, once IWC was outperformed by all other methods, and is even more evident when classifying AndroZoo, given that IWC lost the ability to detect new malware as time goes by. In this case, ADWIN with retraining was able to outperform all the other methods again, even a closely related work (DroidEvolver (Xu et al., 2019)), giving us insight that this is a valid method to be used in practice. In addition, even more important than just using a drift detector, reconsidering the classifier's feature set is required to completely overcome concept drift. According to experiment 5.4, we can infer that retraining the classifier and feature extraction models every time a drift occurs (using the data stream pipeline we proposed) is better than just updating only the classifier. This implies that not only the representation of malware becomes obsolete as time goes by, but also the vocabulary used to build them. It means that every textual attribute used by applications can change, i.e., new API calls, permissions, URLs, for example, emerge, requiring a vocabulary update, i.e., it indicates that discovering new specific features might help in increasing detection rates. Thus, we conclude that malware detection is not just a concept drift problem, but also in essence a feature drift detection problem (Barddal et al., 2017).

Our solution in practice. To implement our solution in practice, we need to model the installation, update, and removal of applications as a stream. This can be done by considering the initial set of applications installed in a stock phone as ground truth and thus subsequent deployment of applications as the stream. To reduce the delay between the identification of a drift point and the classifier update, ideally, multiple classifiers (the current one and the candidates to replace it) should be running in parallel. However, this parallel execution is too much expensive to be performed on endpoints. Therefore, we consider that the best usage scenario for our approach is its deployment on the App's Stores distributing the applications. Therefore, each time a new application is submitted to an App Store, it is verified according to our cycle. To cover threats distributed by alternative markets, this concept can be extended to any centralized processing entity. For instance, an AV can upload the suspicious application to a cloud server and perform its checks according to our cycle. To speed up the upload process, the AV can make the current feature extractor available to the endpoint so as the endpoint does not need to upload the entire application but only its feature. In this scenario, the AV update is not composed of new signatures, but of new feature extractors.

Drift and evolution are common problems in malware detection. Despite being more complex, (more apps and malware families) all characteristics present in DREBIN are drastically shown in AndroZoo, evidencing that feature drift, concept drift, and evolution are present in malware detection problems in practice and not only in a single dataset. This suggests that AV companies should keep enhancing their models and knowledge database constantly.

Limitations and future work. One of the limitations of our approach is that it relies on ground truth labels that may be available with a certain delay, given that known goodware and malware are needed in order to train and update the classifier (as any AV). Thus, future researches to reduce these delays are an important step to improve any ML solution that does not rely on their own labels, such as DroidEvolver (Xu et al., 2019) (outperformed by our approach).

It is important to note that our work considers only the attributes vectors provided by DREBIN (Arp et al., 2014) and AndroZoo (Allix et al., 2016) datasets' authors. Due to this fact, we were unable to consider other types of attributes, such as API semantics (Zhang et al., 2020) or behavioral profiling (dynamic analysis) (Cai, 2020). Therefore, we cannot directly compare our work with theirs, due to different threat models. To compare different attributes using our data stream pipeline under the same circumstances, it is necessary to download all the APKs from both datasets. This is left as future work.

Finally, we make our data stream learning pipeline available as an extension of `scikit-multiflow` (Montiel et al., 2018), aiming at encouraging other researchers to contribute with new strategies that address feature and concept drift.

7. Conclusion

In this article, we evaluated the impact of concept drift on malware classifiers for Android malware samples to understand how fast classifiers expire. We analyzed ≈480 K sample Android apps from two datasets (DREBIN and AndroZoo) collected over nine years (2009–2018) using two representations (Word2Vec and TF-IDF), two classifiers (Adaptive Random Forest and Stochastic Gradient Descent classifier) and four drift detectors (DDM, EDDM, ADWIN, and KSWIN). Our results show that resetting the classifier only after changes are detected is better than periodically resetting it based on a fixed window length. We also point the need to update the feature extractor (besides the classifier) to achieve increased detection rates, due to new features that may appear over time. This strategy was the best in all scenarios presented, even using a complex dataset, such as AndroZoo. The implementation is available as an extension to `scikit-multiflow` (Montiel et al., 2018).[3] Our results highlight the need for developing new strategies to update the classifiers inherent to AVs.

CRediT authorship contribution statement

Fabrício Ceschin: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft. **Marcus Botacin:** Conception and design of study, Analysis and/or interpretation of data. **Heitor Murilo Gomes:** Analysis and/or interpretation of data, Writing – review & editing. **Felipe Pinagé:** Analysis and/or interpretation of data, Writing – review & editing. **Luiz S. Oliveira:** Conception and design of study, Writing – original draft, Writing – review & editing. **André Grégio:** Conception and design of study, Writing – original draft, Writing – review & editing.

[3] https://github.com/fabriciojoc/scikit-multiflow.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link to data is available in the paper.

Acknowledgment

Fabrício would like to thank the Coordination for the Improvement of Higher Education Personnel (CAPES, PhD Scholarship, process 88882.382195/2019-01). Marcus would like to thank the the Brazilian National Counsel of Technological and Scientific Development (CNPq, PhD Scholarship, process 164745/2017-3). All authors approved the version of the manuscript to be published.

Research ethics

We further confirm that any aspect of the work covered in this manuscript that has involved human patients has been conducted with the ethical approval of all relevant bodies and that such approvals are acknowledged within the manuscript.

References

Allix, K., Bissyandé, T. F., Klein, J., & Le Traon, Y. (2016). AndroZoo: Collecting millions of android apps for the research community. In *Int. conf. on min. soft. rep.*.

Anderson, H. S., Kharkar, A., Filar, B., Evans, D., & Roth, P. (2018). Learning to evade static PE machine learning malware models via reinforcement learning. arXiv:1801.08917.

Android (2016). Android 7.0 behavior changes. https://tinyurl.com/yxlpc4gb.

Android (2018a). Accountmanager. https://tinyurl.com/ybsdz76e.

Android (2018b). Device admin deprecation. https://tinyurl.com/yygfkc3m.

Android (2018c). Smsmanager. https://tinyurl.com/y3sz4zzw.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2014). DREBIN: Effective and explainable detection of android malware in your pocket. In *NDSS*.

Bach, S. H., & Maloof, M. A. (2008). Paired learners for concept drift. In *IEEE int. conf. on data mining*.

Baena-Garćıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., & Morales-Bueno, R. (2006). Early drift detection method.

Barddal, J. P., Gomes, H. M., Enembreck, F., & Pfahringer, B. (2017). A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*.

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *SIAM int. conf. on data mining*.

Cai, H. (2018). A preliminary study on the sustainability of android malware detection. arXiv:1807.08221.

Cai, H. (2020). Assessing and improving malware detection sustainability through app evolution studies. *ACM Transactions on Software Engineering and Methodology*, *29*(2), http://dx.doi.org/10.1145/3371924.

Cai, H., & Jenkins, J. (2018). Towards sustainable android malware detection. In *ICSE '18, Proceedings of the 40th international conference on software engineering: companion proceeedings* (pp. 350–351). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3183440.3195004.

Ceschin, F., Pinage, F., Castilho, M., Menotti, D., Oliveira, L. S., & Gregio, A. (2018). The need for speed: An analysis of Brazilian malware classifers. *IEEE Security & Privacy*.

Chang, J., Venkatasubramanian, K. K., West, A. G., & Lee, I. (2013). Analyzing and defending against web-based malware. *ACM Computing Surveys*, *45*(4), http://dx.doi.org/10.1145/2501654.2501663.

Cimpanu, C. (2018). Google restricts which android apps can request call log and sms permissions. https://tinyurl.com/y3y5gfhx.

CTONetworks (2017). Malware infections grow 4X in just one quarter. https://tinyurl.com/yyyugke7.

Deo, A., Dash, S. K., Suarez-Tangil, G., Vovk, V., & Cavallaro, L. (2016). Prescience: Probabilistic guidance on the retraining conundrum for malware detection. In *Proceedings of the ACM workshop on artificial intelligence and security*.

FBReader (2018). For old android devices. https://tinyurl.com/y2odlk9w.

Fergus Halliday (2018). What are the most common kinds of android malware? https://tinyurl.com/wcmr9m2.

Ficco, M. (2022). Malware analysis by combining multiple detectors and observation windows. *IEEE Transactions on Computers*, *71*(6), 1276–1290. http://dx.doi.org/10.1109/TC.2021.3082002.

Fu, X., & Cai, H. (2019). On the deterioration of learning-based malware detectors for android. In *2019 IEEE/ACM 41st international conference on software engineering: companion proceedings (ICSE-Companion)* (pp. 272–273). http://dx.doi.org/10.1109/ICSE-Companion.2019.00110.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*.

Gandotra, E., Bansal, D., & Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*.

Gibert, D., Mateu, C., & Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*.

Hamandi, K., Elhajj, I. H., Chehab, A., & Kayssi, A. (2012). Android SMS botnet: A new perspective. In *Int. symp. on mobility management and wireless access*.

Hurier, M., Suarez-Tangil, G., Dash, S. K., Bissyandé, T. F., Le Traon, Y., Klein, J., & Cavallaro, L. (2017). Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *Int. conf. on min. soft. rep.*.

Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., & Cavallaro, L. (2017). Transcend: Detecting concept drift in malware classification models. In *USENIX security symposium*.

Kantchelian, A., Afroz, S., Huang, L., Islam, A. C., Miller, B., Tschantz, M. C., Greenstadt, R., Joseph, A. D., & Tygar, J. D. (2013). Approaches to adversarial drift. In *ACM workshop on A.I. and security*.

Luo, W., Xu, S., & Jiang, X. (2013). Real-time detection and prevention of android SMS permission abuses. In *Int. W. on sec. in embedded sys. and smartphones*.

Masud, M. M., Al-Khateeb, T. M., Hamlen, K. W., Gao, J., Khan, L., Han, J., & Thuraisingham, B. (2008). Cloud-based malware detection for evolving data streams. *ACM Transactions on Management Information Systems*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. CoRR, arXiv:1310.4546.

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*.

Narayanan, A., Yang, L., Chen, L., & Jinliang, L. (2016). Adaptive and scalable android malware detection through online learning. In *IJCNN*.

Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E. D., Ross, G., & Stringhini, G. (2019). MaMaDroid: Detecting android malware by building Markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security*, *22*(2), http://dx.doi.org/10.1145/3313391.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., & Cavallaro, L. (2018). TESSERACT: eliminating experimental bias in malware classification across space and time. CoRR, arXiv:1807.07838.

Pinage, F. A., dos Santos, E. M., & da Gama, J. a. M. P. (2016). Classification systems in dynamic environments. *WIREs: Data Mining and Knowledge Discovery*.

Raab, C., Heusinger, M., & Schleif, F.-M. (2020). Reactive soft prototype computing for concept drift streams. *Neurocomputing*, *416*, 340–351. http://dx.doi.org/10.1016/j.neucom.2019.11.111.

Rossow, C., Dietrich, C., & Bos, H. (2013). Large-scale analysis of malware downloaders. In U. Flegel, E. Markatos, & W. Robertson (Eds.), *Detection of intrusions and malware, and vulnerability assessment, Berlin, Heidelberg*.

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*.

Sarma, B. P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., & Molloy, I. (2012). Android permissions: A perspective combining risks and benefits. In *Symp. on access control models and technologies*.

SecurityVentures (2018). Global ransomware damage costs predicted to exceed $8 billion in 2018. https://tinyurl.com/y499nvsh.

Singh, A., Walenstein, A., & Lakhotia, A. (2012). Tracking concept drift in malware families. In *Proceedings of the ACM workshop on security and artificial intelligence*.

StatCounter (2018). Operating system market. https://tinyurl.com/tykvtqm.

VirusTotal (2018). Online malware and URL scanner. https://virustotal.com/.

Xu, K., Li, Y., Deng, R., Chen, K., & Xu, J. (2019). DroidEvolver: Self-evolving android malware detection system. In *2019 IEEE EuroS&P*.

Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M., & Yang, M. (2020). Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware. In *CCS '20, Proceedings of the 2020 ACM SIGSAC conference on computer and communications security* (pp. 757–770). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3372297.3417291.