An Experimental Analysis of Semi-supervised Learning for Malware Detection

Luca Minnei[1,*], Giorgio Piras[1], Angelo Sotgiu[1], Maura Pintor[1], Ambra Demontis[1], Davide Maiorca[1] and Battista Biggio[1]

[1] *University of Cagliari, Italy.*

Abstract

In recent years, the wide use of the Android operating system for mobile devices has encouraged a likewise increasing number of cyber-attackers, which exploit related vulnerabilities to create Android Malware. While these represent a major threat in the security landscape, it has been shown how machine learning algorithms, trained over a collection of goodware and malware data, can effectively detect their presence. However, the domain in which such data lies changes over time due to the evolution of applications, such as software updates or deprecation of API calls, and the amount of malware and goodware examples are typically imbalanced. Hence, while machine-learning detectors are effective solutions, their performance must keep up with domain evolution and class imbalance, which can, however, result in frequent expensive retraining. In this work, we perform a preliminary experimental investigation of semi-supervised learning to retrain machine learning-based malware detectors using pseudo-labels along with a small pool of labeled samples. In detail, we account for class imbalance by considering self-training with class-specific thresholds. Our results show that we improve the classification performances by using approximately 10% of pseudo labels in each re-training round.

Keywords

Semi-Supervised Learning, Android Malware Detection, Concept Drift, Active Learning

1. Introduction

Android is one of the most common operating systems for mobile devices, which, however, has often been associated with numerous vulnerabilities.[1] As in many areas of cybersecurity, attackers exploit these vulnerabilities to target users, and Android is no exception. In fact, over the years, attackers have developed and deployed multiple forms of malware, i.e., malicious applications or software.[2] To address this challenge, Machine Learning (ML) models have shown to be highly effective tools in detecting Android malware against legitimate applications (goodware) [1], and as Android mobile devices have become increasingly popular, these detectors play a now potentially crucial role in safeguarding users' security and privacy. In detail, the ML models used in such systems are trained over a collection of goodware and malware samples, and are then used "in the wild" to detect possible threats.

Despite their wide use, recent work has shown that the detection performances of such ML models is, however, mostly limited to controlled environments (in vitro) where data distribution remains stable, while it can drop dramatically when the detectors operate in real-world settings (in vivo) [2]. In fact, the constantly evolving nature of the applications and the corresponding adaptation of malware threats leads to frequent changes in data distribution, which create substantial challenges for detection. This phenomenon, which is known as "concept drift", causes models to lose their effectiveness as the malware landscape evolves and the data distribution changes over time [3]. To keep up with such changes, an

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT

*Corresponding author.

luca.minnei@unica.it (L. Minnei); giorgio.piras@unica.it (G. Piras); angelo.sotgiu@unica.it (A. Sotgiu); maura.pintor@unica.it (M. Pintor); ambra.demontis@unica.it (A. Demontis); davide.maiorca@unica.it (D. Maiorca); battista.biggio@unica.it (B. Biggio)

0009-0005-1207-3439 (L. Minnei); 0000-0001-8225-6138 (G. Piras); 0000-0003-2100-9517 (A. Sotgiu); 0000-0002-1944-2875 (M. Pintor); 0000-0001-9318-6913 (A. Demontis); 0000-0003-2640-4663 (D. Maiorca); 0000-0001-7752-509X (B. Biggio)

[1] CVE database: https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

[2] AV-TEST (2024) https://www.av-test.org/en/statistics/malware/

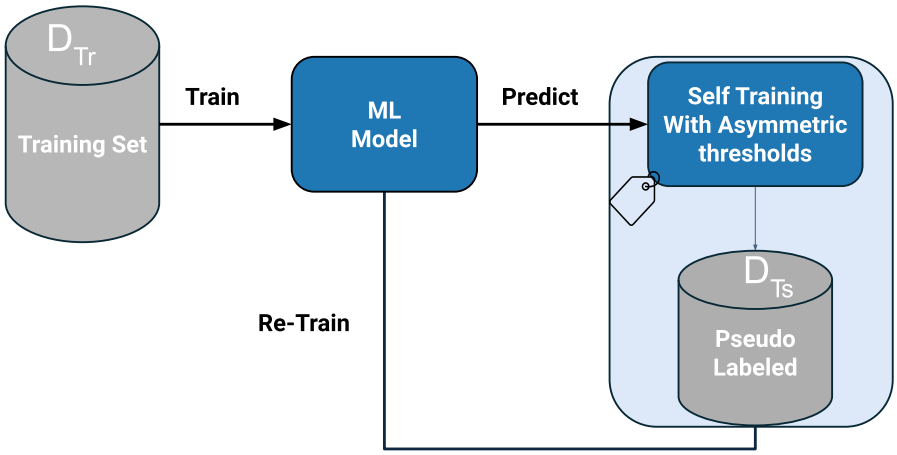

Figure 1: Overview of the investigated approach.

ordinary solution involves frequent retraining of the models on large and manually retrained labeled datasets, which, however, would incur extremely high costs for both retraining and labeling.

In contrast, state-of-the-art approaches adopt different strategies to adapt the model to concept drift, each accompanied by its own set of challenges. For instance, *Continual Learning* allows models to adapt to new malware patterns relying on labeled data to update the model incrementally and ensure accurate detection of evolving threats. However, especially on imbalanced datasets, it risks catastrophic forgetting, a phenomenon where the model loses previously acquired knowledge as it learns from new data [2]. *Semi-Supervised Learning* approaches instead, such as self-training, which uses model predictions as pseudo-labels, reduce reliance on labeled data by leveraging abundant unlabeled apps. However, these require careful management to avoid propagating errors from incorrect pseudo-labels [4]. Finally, *Active Learning* approaches prioritize labeling the most informative samples to reduce labeling costs. For instance, using uncertainty sampling to label the samples improves efficiency, although it requires precise sample identification to maximize impact and minimize additional labeling costs [2]. Hence, while each of these methods proposes interesting approaches, each has its own set of drawbacks that limit the efficacy of a standalone implementation as a solution.

In this work, we thus investigate the use of semi-supervised learning (SSL) to tackle the challenges of updating malware detection models in ever-evolving environments. In detail, we use Self-Training (ST), a subset of SSL, where we leverage asymmetric thresholds to prioritize malware samples and increase the effectiveness of the pseudo-labeling process. In addition, we evaluate an extension of ST to Active Learning (AL) techniques to improve the re-training process by incorporating small batches of labeled samples, and analyze its effectiveness. Our results show that the asymmetric thresholds self-training method outperforms traditional SSL techniques, achieving robust F1 scores even without additional labeled data from AL. By using approximately 10% of pseudo-labels per retraining phase, the method significantly reduces dependence on manual labeling while maintaining high accuracy, demonstrating its efficiency for scalable Android malware detection.

2. Background

We start our discussion by providing the background on Android applications and malware in Sect. 2.1, and then present the related machine-learning techniques for detecting Android malware in Sect. 2.2. We finally conclude by discussing the issues of concept drift in Sect. 2.3.

2.1. Android OS Applications

Applications for the Android Operating System are packaged and installed using Android Application Package (APK) files. These files, with a `.apk` extension, serve as archives containing all the components required for the application, including source code, resources, assets, and metadata.[3] A key element of an APK is the `AndroidManifest.xml` file, which provides essential information for the operating system to install and manage the app, such as the app's name, version, required permissions, and main components. The APK may also include one or more `classes.dex` files containing the app's Java or Kotlin source code compiled into Dalvik bytecode; a `res` directory housing various `.xml` resource files for user interfaces, constant strings, multimedia, and more; a `lib` directory with architecture-specific compiled code, such as native C/C++ libraries; an `assets` folder for general-purpose files; and a `META-INF` directory that stores metadata like certificates and signatures.

Android Malware. The open and flexible nature of the Android operating system makes it an attractive target for malicious actors. Android malware often appears as a legitimate `.apk` file, taking advantage of known vulnerabilities, weak permission controls, and unauthorized app stores. Once inside, it can manipulate core components such as `AndroidManifest.xml`, request unwarranted permissions, or alter `classes.dex` files to insert harmful functionality. After installation, such malware may collect personal information, encrypt files for ransom, display intrusive advertisements, or masquerade as trusted applications. By exploiting the system's packaging, installation, and execution model, Android malware effectively turns the platform's strengths into its own opportunities for malicious activity.

2.2. Machine Learning-based Android Malware Detection

The vast number of applications and malware samples released each year necessitates a high level of analysis and detection efficiency. Traditional signature-based methods, that use the malware signature, are ineffective against sophisticated malware that employs techniques such as obfuscation, app repackaging, dynamic code loading, encryption, and malware dropping, and can easily change the signature without affecting the malware functionality [5]. Moreover, these methods struggle with previously unknown threats and require constant updates to remain effective. To address these limitations, machine learning has emerged as a promising approach for Android malware detection, leveraging features extracted through static [1, 6], dynamic [7, 8], or hybrid analysis [9]. While dynamic analysis provides insights into application behaviour and can prevent evasion tactics, static analysis remains important due to its efficiency and low computational overhead.

Drebin. In this study, we build upon the well-known Drebin malware detection framework [1], which utilizes features extracted statically from Android application files. Drebin defines eight feature sets derived from two key sources: the `AndroidManifest.xml` and `classes.dex` files. These features are summarized in Table 1.

Table 1
Feature sets and their descriptions categorized by manifest and dexcode.

	Feature sets
manifest	S_1: Hardware components
	S_2: Requested permissions
	S_3: Application components
	S_4: Filtered intents
dexcode	S_5: Restricted API calls
	S_6: Used permission
	S_7: Suspicious API calls
	S_8: Network addresses

[3] https://developer.android.com/guide/topics/manifest/manifest-intro

The feature sets S_1, S_2, S_3, and S_4 are extracted from the `AndroidManifest.xml` file, while S_5, S_6, S_7, and S_8 are obtained from the `classes.dex` files. In our approach, these features, represented as text, are transformed into numerical data using a Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer. The TF-IDF vectorizer V converts the extracted features from a dataset D into a d-dimensional vector, where each index corresponds to a specific feature. The value at each index represents the TF-IDF weight of the corresponding feature, reflecting its importance within the dataset. This method considers not only the frequency of a term within a specific document (term frequency, TF) but also its rarity across the entire dataset (inverse document frequency, IDF). By assigning higher importance to terms that are frequent in a given document but rare across the dataset, the TF-IDF approach highlights the most informative features for analysis.

Detector Performance Over Time. Although Drebin and other machine learning-based detectors have demonstrated impressive performance, subsequent studies have revealed that their evaluations often neglected the temporal evolution observed in both legitimate and malicious applications [10]. The introduction of the new Android OS version introduces new functionalities and deprecates or removes others. That leads to changes in the previously defined features, rendering some features obsolete and thereby having no impact on the classification outcome.

These shifts violate the so-called independent and identically distributed (i.i.d.) assumption between training and test datasets in traditional learning-based approaches, necessitating temporally-aware evaluations. Rather than using random splits, data should be partitioned chronologically, training models on older samples and testing on newer ones. Previous research shows that model performance under such temporally-aware evaluations declines significantly over time [10]. Therefore, it is essential to develop new techniques that improve classifiers' robustness to data drift or detect when they become obsolete.

2.3. Countering Concept Drift

While the detectors' performances have been shown to deteriorate over time, multiple solutions have been proposed to address such issues, as described in this section.

Semi-Supervised Learning. Semi-supervised learning (SSL) is used to address the challenges posed by data drift, specifically the scarcity of labeled data. SSL leverages the large amounts of unlabeled data available in malware detection by generating pseudo-labels for unlabeled samples. These pseudo-labels, created using the model's predictions, are then incorporated into the training process, expanding the dataset without additional labeling costs. A commonly used SSL approach is **self-training**, where the model iteratively retrains itself on the most confidently pseudo-labeled samples. Enhancements, such as the use of **asymmetric thresholds** to prioritize the inclusion of malware samples, further improve SSL's effectiveness by ensuring the retraining process targets the most critical and variable aspects of the malware distribution [11]. While recent work has shown the effectiveness of SSL for malware detectors, these approaches often involve costly updates relying on ensembles of models [4] or are limited to neural network architectures [12]. We instead investigate model-agnostic SSL techniques, focusing on the popular Drebin approach features.

Active Learning. Active learning (AL) is a technique designed to optimize the labeling process by selectively querying the most informative samples from the dataset. In the context of malware detection, AL is particularly useful for addressing data drift, as it allows the model to be updated incrementally with minimal manual labeling effort. Common strategies in AL include **uncertainty sampling**, which prioritizes samples where the model has the least confidence in its predictions, and **random sampling**, which serves as a baseline by selecting a random subset of samples. These approaches ensure that labeling resources are focused on the most impactful data points, thereby enhancing the model's ability to adapt to changes in the data distribution over time [13].

3. Self Training with Asymmetric Thresholds

In this section, we first describe, in Sect. 3.1, the semi-supervised learning method employed to efficiently re-train the model, focusing on addressing the challenges posed by concept drift. Then, in Sect. 3.2, we describe the approach used to address the class-imbalance between malware and goodware.

3.1. Yarowsky Algorithm

The base algorithm, called Self-Training (ST) [14], operates by leveraging the most confidently predicted samples as pseudo-labels, which are then used to update the model during the re-training phase. This approach eliminates the need for additional labeled samples. However, it often proves insufficient for re-training the classifier, as it tends to favour the majority class (goodware), thereby exacerbating class imbalance. This limitation arises because the highest-confidence predictions are more likely to belong to the majority class, reducing the focus on the minority class (malware). To address these issues, we experiment with an alternative method that uses asymmetric thresholds to prioritize the minority class during the selection of pseudo-labels.

3.2. Asymmetric thresholds

We address class imbalance through asymmetric thresholds, where we prioritize the selection of malware samples over goodware. Unlike traditional approaches in fact, that use uniform thresholds for all classes, this method assigns distinct thresholds for goodware and malware. Specifically, the algorithm calculates two separate lists of predicted scores: one for samples classified as malware by the model and another for those classified as goodware. For the malware list, a lower threshold is set, allowing a greater number of malware samples to be selected. Conversely, for the goodware list, a higher threshold is applied to limit the number of goodware samples included.

This strategy ensures a higher proportion of malware samples are incorporated, while minimizing the inclusion of goodware samples. Since the initial training dataset already contains a substantial amount of goodware, and recent studies suggest goodware tends to exhibit fewer changes over time, prioritizing goodware selection contributes less to improving the model [15]. The rationale for this approach is that, while malware samples are less frequent, they hold significantly greater value in enhancing the classifier's performance. By lowering the threshold for malware and raising it for goodware, the model focuses on malware instances, improving its ability to detect emerging threats while avoiding overfitting to the abundant but less informative goodware samples. To counteract possible inaccuracies of the labeling process, which is common for SSL algorithms, we additionally evaluate the integration of active learning strategies and analyze the performances with respect to SSL approaches.

4. Experiments

In this section, we first describe the experimental setup used to validate the experiment. Then, we present the results obtained from the experiments and provide insights gained from the analysis.

4.1. Experimental Setting

In this subsection, we describe the experimental setup designed to evaluate the proposed methodology. The setup simulates the real-world scenario, concentrating on the challenges posed by data drift and imbalanced datasets.

Dataset. The datasets are taken from the Android malware detection competition hosted in the ELSA Cybersecurity Use Case.[4] The applications are sampled from the AndroZoo [16] collection of Android Applications, which contains (at the time of writing) over 24 million samples collected from different sources. The sampling is performed based on analysis reports from VirusTotal, from which a timestamp

[4] https://github.com/pralab/elsa-cybersecurity

(from the `first_submission_date` field) and a binary label are extracted. A negative label is assigned to samples that have no detections from the VirusTotal antimalware engines, whereas a positive label is assigned to samples that are detected by at least 10 engines and can be uniquely assigned to a malware family by the *avclass* tool[5], effectively discarding any grayware application that has less than 10 detections or uncertain label. Moreover, a proportion of $9:1$ between legitimate and malware samples is kept, as suggested in previous works [10]. We rely on the provided training set and the test sets of the deployed "Track 3: Temporal Robustness to Data Drift", consisting of a total of $137,500$ samples, of which $123,750$ goodware and $13,750$ malware, sampled between January 2017 and June 2022. Specifically, the initial training set is composed of 75.000 samples collected between January 2017 and December 2019. The subsequent test sets, each spanning a three-month period (one quarter), are sampled starting from January 2020 till June 2022, resulting in a total of 10 distinct test sets.

Models. The experiments in this study were conducted using a Calibrated Linear Support Vector Machine (SVM) model. To enable probability calibration, we employed the `CalibratedClassifierCV` class from Scikit-learn. Calibration is particularly important when dealing with imbalanced data, as models trained on such datasets often produce biased or poorly calibrated probability estimates, favoring the majority class. By calibrating the model, we aim to improve the reliability of the predicted probabilities, ensuring they better reflect the actual likelihoods and enhancing decision-making in downstream tasks. After extensive testing, we selected isotonic regression as the optimal calibration method, setting the number of folds in the inner cross-validation process to 10. This configuration yielded the best performance in terms of calibration accuracy and overall model effectiveness. For the Linear SVC model, we utilized the Hinge Loss function with a regularization parameter of $C = 0.1$. This specific configuration provided a favorable balance between model complexity and generalization, as determined by our experimental results. Additionally, we used the TF-IDF Vectorizer technique for feature tokenization, described in Sect. 2, which is implemented in the Scikit-learn library.

Testing over Time. To support the claim of this study, we trained the model using the dataset from the first three years (referred to as D_{TR}). The remaining data was then divided into consecutive sets covering 3-month periods (quarters). The subsequent quarters were used as D_{TSi}. This method allows us to track changes in the data across each test set and assess the model's performance in the context of temporal drift.

4.2. Experimental Results

In this subsection, we present the results of our experiments on the proposed methodology. The results are analyzed in the context of temporal drift, showing the efficacy of semi-supervised learning techniques and the results of integration with active learning.

Evaluation. The baseline plots in this study will serve as reference points for evaluating the effectiveness of the implemented methods. Baseline plots are calculated for each different setting. **The worst-case scenario, or lower bound,** represents a situation where the model is never retrained over time. This acts as the lower bound because if a method yields results below this curve, it is deemed ineffective; in such cases, doing nothing would be a better option. On the other hand, **the best-case scenario, or upper bound,** represents the optimal outcome where the model is continuously retrained with the correct labels. This upper bound reflects a scenario where the model is regularly updated with new, accurate labels every quarter. Ideally, an effective experiment should yield results that fall between the lower and upper bounds, with a tendency toward the upper bound.

Self-Training Phase. The results from the self-training phase are summarized in a plot that showcases the F1 scores on the y-axis, reflecting the performance of various semi-supervised learning (SSL) techniques, with the quarters displayed on the x-axis. Each curve in the plot corresponds to a specific SSL method: the green curve indicates the asymmetric thresholds SSL method, while the red, purple, and brown curves represent the Scikit-learn SSL techniques. The figures allow for a comparison of these methods under two scenarios: one with no labeled samples from active learning (0 labeled samples) and

[5] https://github.com/malicialab/avclass

another that includes 200 labeled samples obtained through active learning with random sampling. We in fact integrate with Active Learning to simulate a more realistic scenario where the cost of labeling is high, and the model's performances are expected to improve. Visual comparisons for the TF-IDF vectorizer are illustrated in Figure 2 (for 0 labeled samples) and Figure 3 (for 200 labeled samples).

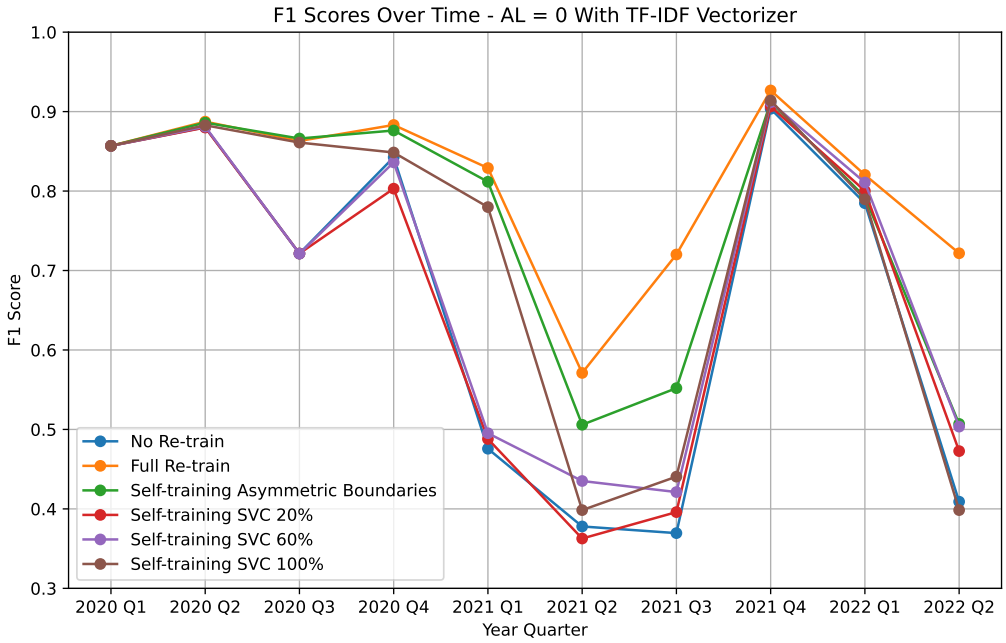

Figure 2: F1 scores compared between the different models using no labels from the active learning technique, using the TF-IDF vectorizer.

From our analysis of the plots, we can highlight two key results. First, the asymmetric thresholds SSL method, represented by the green curve, consistently outperforms the Scikit-learn SSL techniques, regardless of the presence of active learning samples. Second, the performance of the asymmetric method remains stable when transitioning from 0 to 200 labeled samples added in the re-training phase, indicating that the addition of labeled samples through active learning does not enhance its effectiveness. Conversely, the Scikit-learn SSL methods, illustrated by the red, purple, and brown curves, show a slight improvement in performance as we move from 0 to 200 labeled samples. Nevertheless, even with these improvements, these methods still have worse performance than the asymmetric thresholds SSL method in both scenarios.

5. Conclusions and Future Work

This study explores the challenges of retraining classifiers to manage evolving threats while minimizing the use of labeled samples. It particularly focuses on semi-supervised learning (SSL) methods combined with active learning (AL) techniques. Among the various approaches evaluated, the asymmetric thresholds SSL method demonstrated the most robust performance over time. This method not only achieved better performance consistently but also utilized only 10% of the samples during the retraining phases, which included both pseudo-labels and selected labels from the AL process. It effectively prioritized malware samples and avoided selecting redundant, high-confidence goodware samples.

The findings suggest that the high performance of the asymmetric technique is closely linked to the prioritization of malware samples. Although malware instances are less frequent, they have a greater impact during the model update phase. This likely stems from the fact that malware evolves more rapidly [15], enabling it to contribute more effectively to updates in the classifier compared to goodware

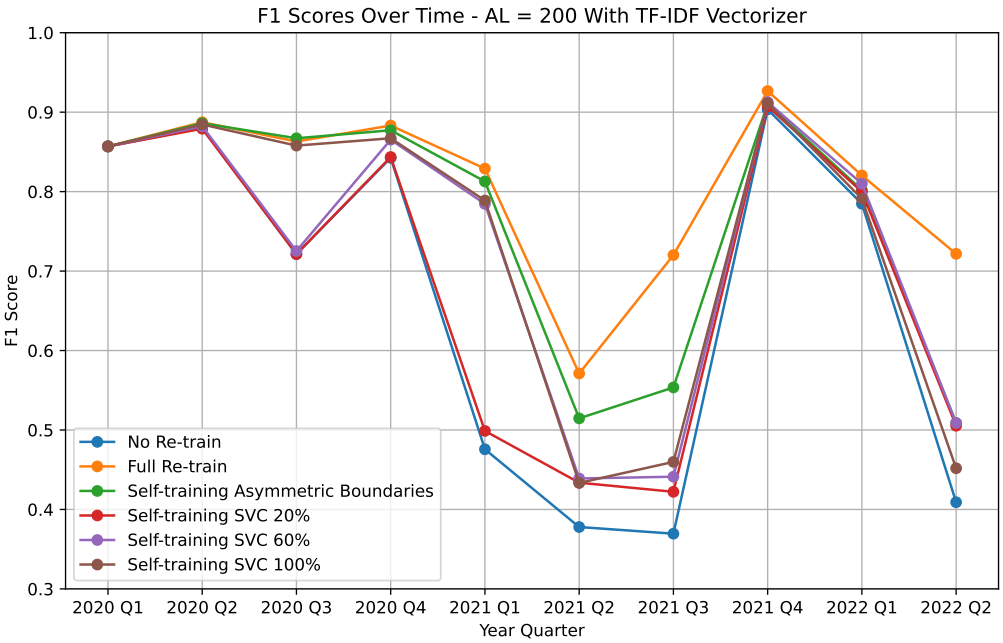

Figure 3: F1 scores compared between the different models using 200 labels from the active learning technique. Using the TF-IDF Vectorizer.

samples. The second insight is that employing random selection within active learning did not improve the performance of semi-supervised learning (SSL), particularly when using fewer than 200 labeled samples. This limitation arises from the imbalanced dataset, which often resulted in the selection of majority-class goodware samples that contributed little to the classification process.

These preliminary results appear to be a promising avenue for further exploration. Future directions include trying this approach with different model retraining techniques. It may also be beneficial to consider alternative approaches beyond SSL, such as contrastive learning techniques or neural networks, which may offer better performance. Additionally, experimenting with more advanced active learning techniques, such as using uncertain samples for labeling, could help identify the most informative data points, while requiring fewer labels. This strategy can potentially enhance model performance and resilience, especially in imbalanced datasets.

Acknowledgments

This work was partially supported by: project SERICS (PE00000014), FAIR (PE00000013, CUP: J23C24000090007) and SETA (PNRR M4.C2.1.1 PRIN 2022 PNRR, Cod. P202233M9Z, CUP F53D23009120001, Avviso D.D. 1409 14.09.2022) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, C. Siemens, Drebin: Effective and explainable detection of android malware in your pocket., in: Ndss, volume 14, 2014, pp. 23–26.

[2] Y. Chen, Z. Ding, D. Wagner, Continuous learning for android malware detection, in: 32nd USENIX Security Symposium (USENIX Security 23), USENIX Association, Anaheim, CA, 2023, pp. 1127–1144. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/chen-yizheng.

[3] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Weiss, Learning under concept drift: A review, IEEE Transactions on Knowledge and Data Engineering 31 (2020) 2346–2363. URL: https://ieeexplore.ieee.org/document/8496795. doi:10.1109/TKDE.2018.2876857.

[4] Z. Kan, F. Pendlebury, F. Pierazzi, L. Cavallaro, Investigating labelless drift adaptation for malware detection, in: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, AISec '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 123–134. URL: https://doi.org/10.1145/3474369.3486873. doi:10.1145/3474369.3486873.

[5] V. Rastogi, Y. Chen, X. Jiang, Droidchameleon: Evaluating android anti-malware against transformation attacks, in: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ACM, New York, NY, USA, 2013, pp. 329–334. doi:10.1145/2484313.2484355.

[6] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, G. Stringhini, Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version), ACM Trans. Priv. Secur. 22 (2019). URL: https://doi.org/10.1145/3313391. doi:10.1145/3313391.

[7] A. Saracino, D. Sgandurra, G. Dini, F. Martinelli, Madam: Effective and efficient behavior-based android malware detection and prevention, IEEE Transactions on Dependable and Secure Computing 15 (2018) 83–97. doi:10.1109/TDSC.2016.2536605.

[8] H. Cai, N. Meng, B. Ryder, D. Yao, Droidcat: Effective android malware detection and categorization via app-level profiling, IEEE Transactions on Information Forensics and Security 14 (2019) 1455–1470. doi:10.1109/TIFS.2018.2879302.

[9] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, J. Hoffmann, Mobile-sandbox: combining static and dynamic analysis with machine-learning techniques, Int. J. Inf. Secur. 14 (2015) 141–153. URL: https://doi.org/10.1007/s10207-014-0250-0. doi:10.1007/s10207-014-0250-0.

[10] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, L. Cavallaro, {TESSERACT}: Eliminating experimental bias in malware classification across space and time, in: 28th USENIX security symposium (USENIX Security 19), 2019, pp. 729–746.

[11] J. E. van Engelen, H. H. Hoos, A survey on semi-supervised learning, Machine Learning 109 (2020) 373–440. URL: https://doi.org/10.1007/s10994-019-05855-6. doi:10.1007/s10994-019-05855-6.

[12] M. T. Alam, R. Fieblinger, A. Mahara, N. Rastogi, Morph: Towards automated concept drift adaptation for malware detection, 2024. URL: https://arxiv.org/abs/2401.12790. arXiv:2401.12790.

[13] F. Cacciarelli, M. Kulahci, Active learning for data streams: A survey, Machine Learning 113 (2024) 45–72. URL: https://link.springer.com/article/10.1007/s10994-023-06454-2. doi:10.1007/s10994-023-06454-2.

[14] D. Yarowsky, Unsupervised word sense disambiguation rivaling supervised methods, in: Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics, ACL '95, Association for Computational Linguistics, USA, 1995, p. 189–196. URL: https://doi.org/10.3115/981658.981684. doi:10.3115/981658.981684.

[15] L. Minnei, H. Eddoubi, A. Sotgiu, M. Pintor, A. Demontis, B. Biggio, Data drift in android malware detection, in: International Conference on Machine Learning and Cybernetics, ICMLC, IEEE, 2024.

[16] K. Allix, T. F. Bissyandé, J. Klein, Y. Le Traon, Androzoo: Collecting millions of android apps for the research community, in: Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, ACM, New York, NY, USA, 2016, pp. 468–471. URL: http://doi.acm.org/10.1145/2901739.2903508. doi:10.1145/2901739.2903508.