A Comparative Analysis of Active Learning Strategies for Android Malware Detection

CRISTIAN MANCA[1], LUCA MINNEI[1], MAURA PINTOR[1],
FABIO BRAU[1] BATTISTA BIGGIO[1]

[1]University of Cagliari, Italy

Abstract:

Android malware detectors increasingly rely on machine learning algorithms that are trained on datasets containing both benign (goodware) and malicious (malware) applications. These detectors have shown excellent results when the training and testing sets are collected over a fixed period. However, recent research indicates that the domain is not static due to ongoing changes in applications, which can lead to a decline in detector performance over time. The most effective solution to maintain the performance of these detectors is to continuously retrain them to update their knowledge. However, labeling data can be costly, as each sample requires analysis by a specialist. One straightforward approach is to use Active learning (AL), implementing techniques that select a subset of the most informative samples to be labeled, and leave the rest unlabeled. Despite its potential, there have been few attempts to compare and evaluate existing AL methods. In our study, we test six benchmark strategies to evaluate and compare their effectiveness in the Android malware domain. Our results show that 10% of the data is labeled using any of these methods is enough to achieve detector performance closely matching that of a fully supervised model. This confirms that AL can effectively counter concept drift while keeping labeling costs to a minimum.

1 Introduction

Android is currently the most popular mobile operating system. Still, its popularity has also made it a prime target for cyberattacks aimed at developing malicious applications that can compromise users' data and security. Detecting malicious Android applications has become a critical task for mobile security. Machine learning (ML) has been widely adopted to automate malware detection, demonstrating excellent performance when trained on large datasets of both benign and malicious applications [?]. However, recent studies [?] have shown that this domain is prone to data drift over time, which causes significant degradation in the performance of ML models when deployed in real-world scenarios. To address this problem, detection models need to be retrained to adapt to the ever-evolving threats. However, acquiring labels for new samples is expensive and requires significant human expertise. Active learning (AL) offers a promising solution by selectively querying labels for the most informative samples, reducing the labeling effort while maintaining high detection performance [?]. Although several AL strategies have been proposed in different domains, few works have systematically evaluated these methods in the context of Android malware detection.

In this paper, we compare several AL strategies provided by the scikit-activeml library[1] in a realistic context affected by temporal drift. We simulate a time-aware evaluation where new applications arrive sequentially in time and analyze the model's ability to adapt using different AL strategies and labeling budgets. Through this analysis, we aim to provide insights into (i) the ability of AL strategies to maintain detection performance in the presence of data drift, and (ii) the comparative effectiveness of different standardized AL strategies in identifying the most suitable approaches for Android Malware detection. Our results show that labeling 10% of the test set with AL strategies can achieve performance comparable to an ideal cumulative retraining scenario, thus significantly reducing annotation costs while maintaining high detection quality.

In the following, we first introduce Android malware detection and the challenge of temporal drift (Sect. 2). We then describe the AL scenario and available strategies

[1]https://github.com/scikit-activeml/scikit-activeml

(Sect. 3). Afterwards, we present our analysis (Sect. 4), followed by conclusions and future research directions (Sect. 5).

2 Machine Learning for Android Security

This section provides essential background information on Android applications and ML-based techniques to detect Android malware. Furthermore, we describe the main issue of data drift for these approaches.

Android Application Structure. Android Applications are packaged and distributed, and installed through Android Application Package (APK) files. These are archive files with a .apk extension that encapsulate all components necessary for installation and execution. An APK includes the application bytecode, compiled from Java or Kotlin in Dalvik format, within one or more classes.dex files. It also contains the AndroidManifest.xml, a file that declares essential metadata such as the app's name, version, permissions, and the main components of the application.

ML-based Android Malware Detection. In this work, we adopt a feature extraction approach inspired by Drebin [?], a static malware detector based on a linear SVM, which extracts a variety of static features from Android applications. Specifically, features such as permissions, components, filtered intents, API calls, and network addresses are derived from the AndroidManifest.xml and the classes.dex files, and represented using a binary vector encoding their presence. In this setting, the machine learning input X consists of binary vectors representing the presence or absence of static features in each application, while the corresponding label y indicates whether the application is benign or malicious.

Impact of Temporal Drift. Despite the initially reported excellent performance of Drebin and similar learning-based detectors, subsequent studies have highlighted that, to properly assess these models in realistic setting, the data must be split chronologically. Specifically, training the data on earlier samples and testing it on later ones revealed significant drops of model performance over time [?].

First, the Android OS introduces new APIs and deprecates older ones, causing changes in the feature space. Furthermore, while new malware families emerge rarely, a large number of variants are continuously developed to evade detection. These variants may exhibit patterns not captured by the training data, resulting in changes in distribution. Therefore, some features may become obsolete, while others may not be detected by static analysis, as they violate the standard assumption of independence and identical distribution of data in the training and test sets, which is the basis of most ML models. This problem can be solved by either designing models robust to temporal drift, or by retraining the models with new and more recent samples. However, this second strategy requires significant labeling cost, therefore being difficult to implement in practice.

3 Active Learning

In this section, we will introduce the theory behind AL, with a focus on the techniques used in our experiment. We will also discuss previous research that has investigated AL in the field of malware detection.

AL is a branch of ML that emphasizes the iterative selection of the most informative unlabeled samples for annotation, based on specific criteria known as the AL strategy. These strategies define which samples are prioritized for labeling, to maximize model improvement while minimizing the number of labeled examples required. AL can be categorized into several subfields based on how and when the samples are selected:

- Pool-based. The model has access to a large pool of unlabeled instances, and the goal is to select the most informative samples to query for labeling;
- Stream-based. Assumes that instances arrive in a continuous stream over time, and decisions about labeling must be made immediately;
- Query Synthesis The learner has complete freedom to label any data point belonging to the input space or for a synthetically generated one;

Among the three categories, pool-based is the most suitable for Android malware detection because, in real-world scenarios, we often deal with large pools of applications whose maliciousness is unknown. At the same time, the number of labels that an analyst can provide is limited, especially within tight time constraints. Therefore, it is essential to identify and select the most informative samples from the pool to maximize the impact of each labeling effort.

3.1 Pool-based Strategies

The AL pool-based strategies distinguish themselves by their definitions of what constitutes an informative sam-

ple. The interpretation of an informative sample can vary depending on the underlying model or the domain in which the technique is applied. Different strategies have been proposed in the literature, each aiming to exploit different properties of the data. In the following, we detail the strategies considered in our evaluation.

Random Sampling. Random Sampling is the simplest form of selection, where instances are selected uniformly at random from the unlabeled data set for labeling. Despite its simplicity, it serves as a baseline for evaluating the effectiveness of more advanced AL strategies.

Margin Sampling. Margin sampling [?] selects the sample for which the difference between the top two predicted class probabilities is the smallest:

$$x^* = \arg\min_x \left(P(y_1|x) - P(y_2|x)\right)$$

A small margin implies that the model is almost equally uncertain between the two classes.

Least-Confident Sampling. Least-Confident sampling [?] selects the instance for wich the model has the least confidence in its prediction:

$$x^* = \arg\min_x P(y^*|x)$$

where: $y^* = \arg\max_y P(y|x)$. It targets samples where the classifier is farthest from a confident decision, focusing on instances that lie close to the decision boundary.

Entropy Sampling. Entropy sampling [?] selects the sample with the highest predictive entropy, defined as:

$$x^* = \arg\max_x -\sum_{i=1} P(y_i|x) \log P(y_i|x)$$

where $P(y_i|x)$ is the predicted probability of class y_i. Entropy measures the total uncertainty of the model across all classes: a higher entropy indicates greater uncertainty.

Expected Average Precision (EAP). EAP [?] estimates, for each sample, the expected improvement in Average Precision (AP) if the sample were labeled and added to the training set:

$$x^* = \arg\max_x \mathrm{EAP}(x), \quad \text{where:}$$

$$\mathrm{EAP}(x) = P(y^* = 1|x^*) \cdot \mathrm{AP}_{new}^{(y^*=1)} + P(y^* = 0|x^*) \cdot \mathrm{AP}_{new}^{(y^*=0)}$$

AP is a more complex evaluation metric than accuracy and can be described as the area under the precision-recall curve. This strategy favors samples likely to improve the model's ranking quality, which is especially useful in unbalanced or detection tasks.

CLustering with Uncertainty-weighted Embeddings (CLUE). CLUE [?] selects uncertain samples representative in the feature space. It optimizes the selection via:

$$x^* = \arg\min_x \left(u(x) - \mathrm{sim}(x, C_k)\right)$$

where $u(x)$ denotes the model uncertainty for sample x, C_k is the centroid of the cluster to which x belongs, and $\mathrm{sim}(x, C_k)$ measures the similarity between x and the cluster center. This balances informativeness (uncertainty) and diversity (input space coverage).

3.2 Active Learning for Android Malware Detection

Building on the characteristics of AL outlined in Sect. 3, its application in the malware detection domain is straightforward. AL is particularly effective at identifying the most informative samples. Since labeling samples can be costly, especially when experts must manually analyze suspicious binaries, this approach is well-suited for enhancing Android malware classifiers while minimizing the need for extensive annotation.

Several recent studies have explored AL techniques for malware detection. ActDroid [?] implements an uncertainty sampling strategy based on prediction margins for streaming Android malware, combined with semi-supervised learning to exploit both labeled and unlabeled data. MalOSDF [?] applies classical uncertainty-based querying to enhance a semi-supervised ensemble classifier operating over opcode slice features, where selection and model update are tightly coupled. MORPH [?] adopts a margin-based active selection scheme tailored to pseudo-labeled samples, dynamically adapting the margin thresholds to account for concept drift. Continuous Learning for Android Malware Detection [?] integrates a custom prioritization function based on contrastive learning to actively select samples for retraining within a continual learning setting. While these works demonstrate the potential of AL, their solutions are primarily customized to particular detection architectures, feature sets, or learning setups. In contrast, our work systematically benchmarks standardized, model-agnostic AL strategies within a realistic Android malware detection setting, promoting reproducibility and broader applicability across different systems.

4 Experiments

This section will explain the experimental setup and the results obtained from our experiments.

Dataset. The dataset used in our experiments is derived from the Android malware detection competition provided as part of the ELSA Cybersecurity Use Case.[2] The applications are sampled from the AndroZoo repository [?], a large-scale collection of Android applications. Each application is labeled based on VirusTotal scan reports: samples with no detections by any antivirus engine are labeled as benign, while those detected by at least 10 engines and unambiguously mapped to a malware family using the avclass tool[3] are labeled as malicious. Samples with ambiguous or insufficient detection evidence are excluded. Following standard practice in the literature [?], the dataset maintains a 9:1 ratio between benign and malicious samples. For our analysis, we rely on the training and test sets provided by the organizers for "Track 3: Temporal Robustness to Data Drift", comprising a total of $137,500$ applications ($123,750$ benign and $13,750$ malware), collected between January 2017 and June 2022.

Although AL typically benefits from dense feature representations, we applied a feature selection step to reduce computational overhead without compromising model performance. Specifically, we retained only the 10,000 most frequent features, ranked according to their frequency of occurrence in the initial training set. This dimensionality reduction significantly accelerates both the model training and AL steps, while maintaining comparable detection performance.

Retraining Phase. To simulate a realistic evaluation under temporal drift, the dataset is split chronologically. The first $75,000$ samples, covering the period from January 2017 to December 2019, are used as the initial training set. The remaining $62,500$ samples, spanning the period from January 2020 to June 2022, are split into 10 time-ordered segments, each containing $6,250$ applications. At each iteration (so approximately every 3 months in the simulation), one quarter is used as a test set to evaluate the model's predictions. After evaluation, AL is applied by selecting new informative samples from the current test quarter and adding them to the training set. Specifically, we test four labeling budgets: 60, 125, 312, and 625 samples, which correspond to approximately 1%, 2%, 5%, and 10% of applications available in each quarter. The resulting augmented training set is then used to retrain the model before proceeding to the next quarter. This setup allows us to evaluate the effectiveness of different AL strategies in fitting the model to a non-stationary data distribution over time.

Models. We use a Linear Support Vector Machine (SVM) classifier with Hinge Loss and $C = 0.1$, consistent with the baseline model provided in the ELSA Cybersecurity Use Case. To ensure compatibility with the scikit-activeml framework, which requires classifiers to conform to specific interface requirements, we implemented a lightweight wrapper around the standard scikit-learn SVM. Since several AL strategies require reliable probability estimates, the SVM was calibrated using a sigmoid calibration method implemented via CalibratedClassifierCV. This calibration step ensures that the decision function outputs can be interpreted as probabilities, enabling uncertainty-based AL strategies. In addition to the SVM, we evaluate a standard Random Forest classifier, configured with 80 trees and a maximum depth of 30. The hyperparameters for both models were selected through manual cross-validation, testing multiple configurations and selecting the ones yielding the best overall performance. Both classifiers operate on binary feature vectors generated using a CountVectorizer fitted on the initial training set.

Performance Evaluation. To evaluate the model's performance, we use the following standard classification metrics:
Precision measures the proportion of correctly predicted positive samples among all samples classified as positive: $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$;
Recall (also known as True Positive Rate) measures the proportion of correctly identified positive samples among all actual positive samples: $\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$;
F1 is the harmonic mean of precision and recall, providing a balanced measure particularly useful under class imbalance: $\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$;
Accuracy measures the proportion of correctly classified samples (both positive and negative) among all samples: $\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$;

Results. For each labeling budget, we compare the AL strategies described in Sect. 3 against two reference baselines: (i) a Naive baseline, where the model is only trained on the initial training set without any updates, and (ii) a Cumulative baseline, where all test samples are incrementally added to the training set without selection (i.e., assuming unlimited labeling capacity). The performance

[2] https://github.com/pralab/elsa-cybersecurity
[3] https://github.com/malicialab/avclass

trends, illustrated in Fig. 1, clearly show that retraining with AL strategies can significantly mitigate the performance degradation caused by temporal drift across both Linear SVM and Random Forest models. Specifically, Uncertainty Sampling consistently outperforms both the Naive baseline and Random Sampling across all budgets. Among these, CLUE demonstrates strong and stable performance across both models, whereas EAP achieves good results with the SVM but struggles with the Random Forest, showing lower F1 scores and greater instability. Notably, when using a labeling budget of 625 samples per quarter, almost all tested AL strategies achieve F1 scores comparable to the Cumulative baseline, thus approaching the ideal scenario with only a fraction of the labeling effort. Although Random Sampling shows some improvement over the naive baseline, it remains significantly less effective than uncertainty-based approaches, highlighting the importance of selecting more informative strategies that are well-suited to the model and the evolution of the data distribution. These observations highlight that AL can achieve high performance over time if the right strategy is chosen, significantly reducing the amount of labeled data needed, and thus the human effort. Furthermore, Tab. 1 provides a detailed summary of the average Precision, Recall, F1-score, and Accuracy achieved by each strategy on different labeling budgets. The results confirm that uncertainty-based methods consistently outperform Random Sampling and the Naive baseline, with much less annotation effort than the Cumulative baseline. Furthermore, we observe that the gap between uncertainty-based strategies and the Naive baseline becomes even more pronounced at higher labeling budgets, highlighting the ability of AL to increasingly exploit informative samples as more labels become available.

5 Conclusions

In this paper, we compare different AL strategies in a realistic time-drift context. Different from most existing approaches, which typically design customized solutions, we systematically evaluate standardized AL methods along with a standard detection model, without introducing domain-specific adaptations. This enables an objective and reproducible comparison of different approaches in the context of Android malware detection. Our results demonstrate that AL significantly mitigates the performance degradation caused by temporal drift. In particular, with a labeling budget of only 10% of the available samples, most of the tested uncertainty-based strategies achieve F1 scores comparable to the ideal cumulative retraining scenario, highlighting the potential of AL to maintain high detection performance while substantially reducing labeling costs. In future work, we plan to extend our evaluation to different malware detection datasets, including non-Android environments, to assess the generalization capabilities of the tested strategies. Furthermore, we aim to explore the integration of AL with semi-supervised learning and continual learning paradigms to further enhance model adaptation over time. Finally, we will investigate novel AL strategies specifically designed to minimize the labeling budget while maintaining robustness against complex and evolving drift patterns.

Acknowledgements

This work has been partly supported by the EU-funded Horizon Europe projects ELSA (GA no. 101070617), Sec4AI4Sec (GA no. 101120393) and CoEvolution (GA no. 101168560); and by the projects SERICS (PE00000014) and FAIR (PE00000013) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

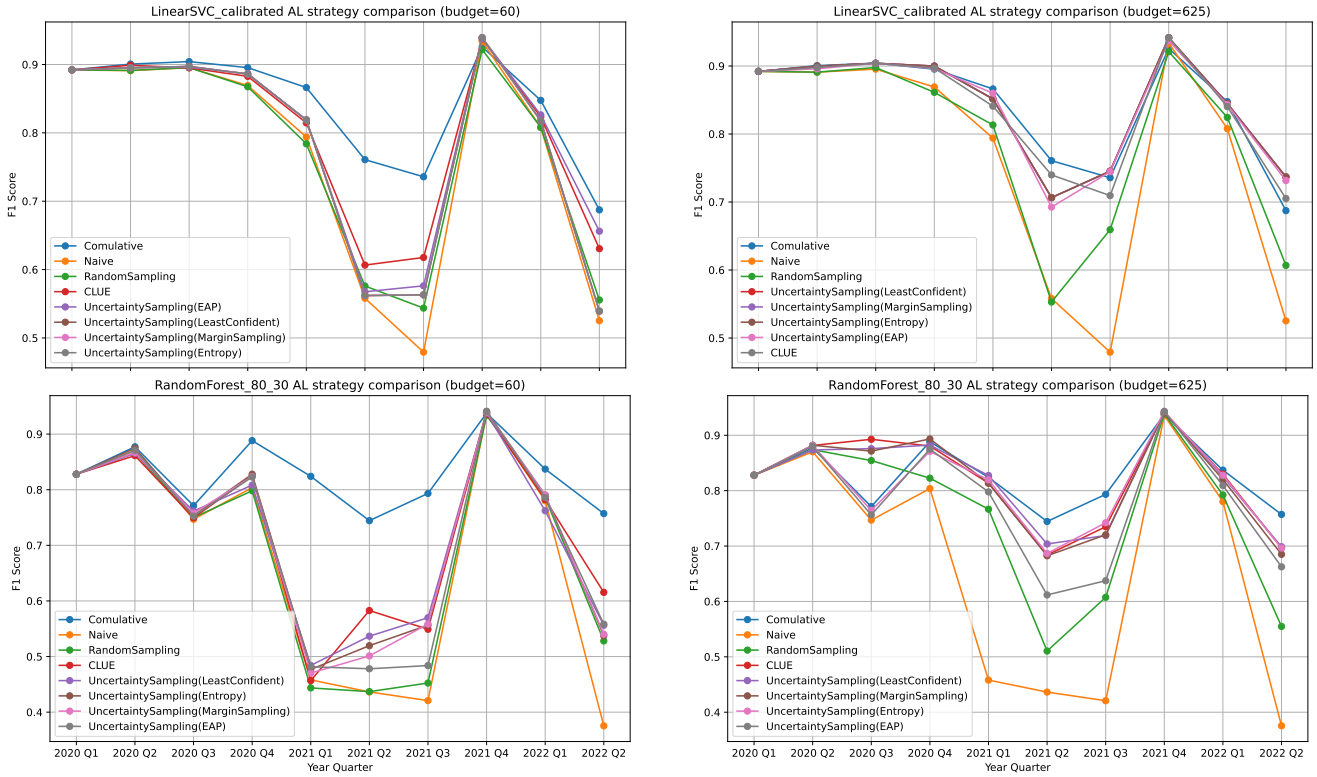

FIGURE 1. Linear SVM and RandomForest comparison curves of F1-score evolution over time using 60 and 625 labeled samples per quarter with different AL strategies.

Strategy	1% Budget				2% Budget				5% Budget				10% Budget			
	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc
Model: Linear SVM																
Naive	0.949	0.667	0.765	0.966	0.949	0.667	0.765	0.966	0.949	0.667	0.765	0.966	0.949	0.667	0.765	0.966
RandomSampling	0.922	0.684	0.774	0.966	0.925	0.696	0.783	0.966	0.926	0.705	0.791	0.967	0.927	0.707	0.792	0.968
Least-Confident	0.959	0.685	0.781	0.968	0.965	0.703	0.799	0.970	0.967	0.735	0.827	0.973	0.965	0.756	0.842	0.974
MarginSampling	0.959	0.685	0.781	0.968	0.965	0.703	0.799	0.970	0.967	0.735	0.827	0.973	0.965	0.756	0.842	0.974
EntropySampling	0.959	0.685	0.781	0.968	0.965	0.703	0.799	0.970	0.967	0.735	0.827	0.973	0.965	0.756	0.842	0.974
EAP	0.963	0.700	0.795	0.969	0.967	0.719	0.814	0.971	0.967	0.734	0.827	0.973	0.964	0.753	0.840	0.974
CLUE	0.961	0.704	0.800	0.969	0.967	0.713	0.809	0.971	0.967	0.734	0.827	0.973	0.965	0.748	0.837	0.974
Cumulative	0.941	0.767	0.842	0.974	0.941	0.767	0.842	0.974	0.941	0.767	0.842	0.974	0.941	0.767	0.842	0.974
Model: Random Forest																
Naive	0.903	0.563	0.666	0.954	0.903	0.563	0.666	0.954	0.903	0.563	0.666	0.954	0.903	0.563	0.666	0.954
RandomSampling	0.913	0.578	0.682	0.955	0.917	0.594	0.698	0.956	0.928	0.638	0.741	0.961	0.925	0.659	0.755	0.963
Least-Confident	0.923	0.603	0.711	0.958	0.932	0.668	0.764	0.964	0.937	0.687	0.783	0.966	0.940	0.734	0.818	0.970
MarginSampling	0.924	0.602	0.708	0.958	0.926	0.628	0.728	0.960	0.940	0.698	0.793	0.967	0.942	0.728	0.814	0.970
EntropySampling	0.930	0.603	0.711	0.958	0.931	0.620	0.725	0.960	0.937	0.693	0.786	0.966	0.941	0.715	0.806	0.969
EAP	0.925	0.594	0.700	0.957	0.927	0.639	0.742	0.961	0.936	0.672	0.772	0.964	0.940	0.682	0.780	0.966
CLUE	0.918	0.615	0.719	0.958	0.929	0.665	0.764	0.963	0.938	0.692	0.788	0.966	0.940	0.735	0.818	0.970
Cumulative	0.938	0.745	0.826	0.971	0.938	0.745	0.826	0.971	0.938	0.745	0.826	0.971	0.938	0.745	0.826	0.971

TABLE 1. Comparison of Precision, Recall, F1 score, and Accuracy across different AL strategies, budgets, for the two implemented models.