TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Dmytro Martynov 201629IVCM

Handling Concept Drift for Mobile Malware Detection

Master's thesis

Supervisor: Hayretdin Bahsi
PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Dmytro Martynov 201629IVCM

Mõistenihke haldamine mobiilpahavara tuvastamisel

Magistritöö

Juhendaja: Hayretdin Bahsi
PhD

Tallinn 2023

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Dmytro Martynov

10.05.2023

Abstract

During the last decade mobile platforms achieved enormous amounts of capabilities previously only available on desktops. Those capabilities combined with the native features of mobile devices have made them increasingly popular. At the same time, the prevalence of mobile platforms makes them a target for the malefactors, who are creating mobile malware. The current counter solutions are vulnerable to the constantly evolving mobile malware, as the statistical properties and behavioral characteristics of malware samples change over time reducing the effectiveness of machine learning based malware detection models. This process is usually referred to as concept drift. This thesis will study handling of concept drift for mobile malware detection based on the Android malware dataset, as Android holds three quarters of mobile devices market share. The experiment will be performed using the periodized datasets, where each subset represents the quarter of a year in the range 2010-2018. The experiment includes two testing scenarios: 1) testing on totally new, previously unseen data only; 2) testing on realistic datasets that also include older samples and keep a portion of malicious samples approximately 10%. In this work there will be a comparison of 1) system calls and permissions as selected features; 2) various data balancing techniques; 3) model updating strategies including retraining and incremental learning.

This thesis is written in English and is 36 pages long, including 6 chapters, 18 figures and 2 tables.

List of abbreviations and terms

PC	Personal Computer
OS	Operation System
NFC	Near Field Communication
APK	Android Package
ML	Machine Learning
RF	Random Forest
SMOTE	Synthetic Minority Over-sampling Technique
ENN	Edited Nearest Neighbours
TN	True Negatives
TP	True Positives
FN	False Negatives
FP	False Positives

Table of contents

List of figures

List of tables

Introduction

Despite the fact that the word “malware” is typically associated with personal computers and laptops, something that runs desktop operating systems, mobile malware has become an incredibly worrying problem. To understand why mobile malware is such a big issue, it is important to familiarize with the stages of evolution of mobile devices, how it became comparable to PC (Personal Computer) in the number of web visits, fulfilling the need of solving internet related tasks previously inherent to desktop.

The history of smartphones is considered to start in the 1990s with the release of IBM Simon Personal Communicator that combined features on a cellphone and personal computer. Since that time other companies like BlackBerry, Ericsson and Nokia have made their own products with various levels of commercial success [1], [2]. The main features of the smartphones were their ability to connect to the internet and install third-party applications. Such description is applicable to nowadays smartphones as well. The next stage for mobile devices is undoubtedly the revolutionary release of the iPhone by Apple in 2007. iPhone incorporated full touch screen display, intuitive OS (Operating System), ergonomic design, laconic appearance, number of built-in applications that are now considered to be essential and introduced App Store that allows easy and quick installation of various applications [3]. The tremendous commercial success of the iPhone transformed the mobile phone industry drastically [4]. Every respectable tech giant company intended to create its own iPhone. And Google was willing to help with its own mobile OS – Android. With Android Google offered an open-source mobile operating system to the tech companies, letting them concentrate on hardware [5]. Android’s availability and customizability made its popularity increase rapidly, surpassing iOS in terms of global market share in the late 2010 – early 2011 and reaching approximately 71% of global market share by March 2023 [6]. (See Figure 1) As Google announces on Google I/O 2021, there are over 3 billion active Android devices in total [7]. Figure 1 illustrates how IOS and Android conquered the market of mobile operating systems and later shaped it to the current situation, where the market is divided 1:3 between Apple's IOS and Google's Android respectively, without any real competitors left.

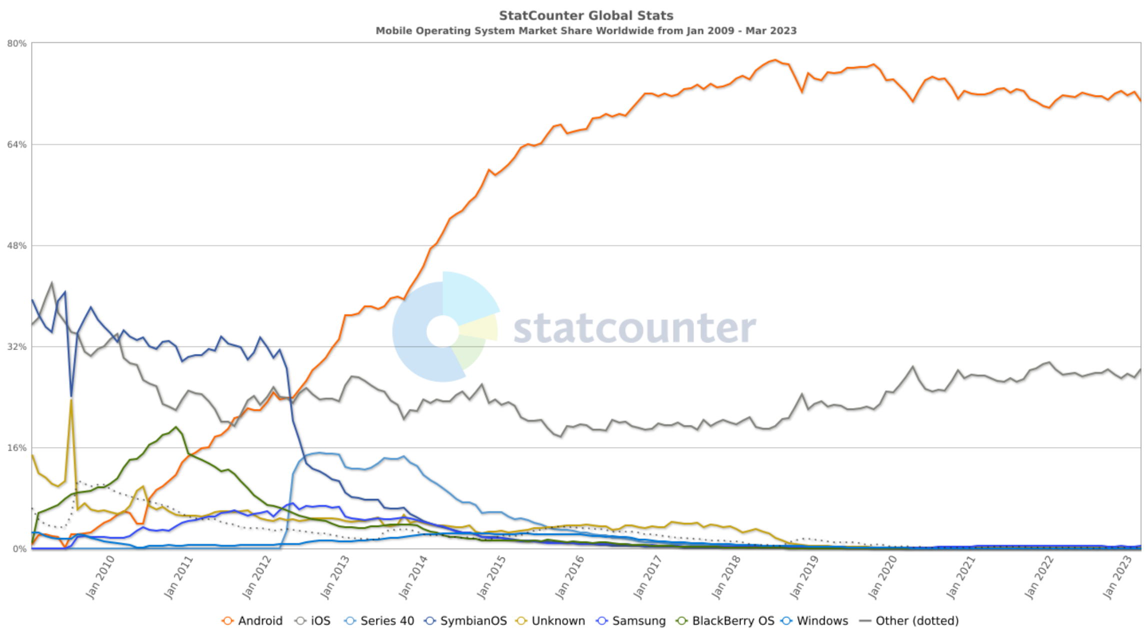


Figure 1. Mobile OS market share worldwide 2009-2023 according to StatCounter [6].

Through the years of evolution, mobile platforms received additional functionality when emails, social media and video calls were introduced to classical calls and SMS, extending communication abilities of mobile devices. The appearance of cameras in the smartphones that over time reached an impressive quality of photos, even managed to supplant point-and-shoot cameras from the amateur photography market [8]. Another essential feature of mobile devices is mobile banking that allows to perform transactions and manage the accounts from a smartphone or tablet. In addition to that, smartphones have even made payment through the device possible using NFC (Near Field Communication), for example, Google Pay in Android and Apple Pay in IOS. In order to increase security, biometric technologies were implemented for user authentication, such as fingerprint sensors, facial, voice and even iris recognition (the first two being the most popular). All these extremely valuable features in combination with portability and user-friendly experience made modern people's lives unimaginable without a smartphone. Therefore, soon after smartphone manufacturers managed to achieve decent quality of their products, the mobile market started to gradually grow catching up with desktops in terms of sales in 2016-2017, and eventually surpass desktops in recent years [9].

The functionalities described above make mobile devices a treasure trove of personal, valuable and sensitive information, but at the same time desirable target for cyber criminals. Malefactors design malicious software to infect mobile devices, steal or encrypt user's data or damage the device itself. According to AV-Atlas almost 1.1 million of new malware appeared in 2022, while the peak growth of new Android malware was in 2016 and 2017 when the number of malware increased by over 6 million samples [10]. AV-Atlas stated that the total number of Android malware reached 33.5 million (those numbers exclude potentially harmful apps) [10]. Common types of Android malware are trojans, designed to steal sensitive information like banking credentials and credit card numbers, adware, that displays annoying adverts through pop-ups and banners, spyware, that monitors various user activities, and ransomware, encrypting user data and asking for a ransom. According to Kaspersky SecureList, top three mobile malware types in 2022 were RiskTool (potentially malicious applications) – 27.4%, Adware – 24%, Trojan – 15.6% [11].

There are many reasons why Android is vulnerable to cyber threats. Android is an open-source operating system that can run on various types of hardware. Android allows installation of APKs (Android Package) from untrusted sources. However, even Google Play Store does not guarantee absolute safety of the applications. In 2023 McAfee disclosed new Android malware called 'Goldoson' that has infiltrated Google Play through 60 legitimate applications with a total number of downloads exceeding 100 million [12]. It became possible despite Google launching its machine learning backed built-in malware defense - Google Play Protect [13]. Another Android problem is that many devices are not updated to the latest version either because the devices are no longer supported or because users ignore the security patches. Thus, by April 2023 the most popular Android version is 11 with 21.1%, next is 12 – 20.9%, and the latest Android 13 is just 20% [14]. In comparison, iOS 16 share is 74.6% [15].

Due to the popularity of Android, in this thesis concept drift of mobile malware will be studied based on the Android malware dataset. Although Android and IOS have their own peculiarities due to the distinctions in operating system architectures, causing differences in infection vectors and malware behavior, it is reasonable to assume that finding of this research can be helpful for studies of IOS malware concept drift.

1.1 Problem Statement and Research Questions

Developed by Google, Android is the most widespread operating system in the mobile segment, reaching almost three-quarter market share. At the same time, the popularity of Android, its open-source nature, possibility to install applications from unauthorized sources avoiding Google Play Store security checks and problems updating devices with the latest security patches make this operating system an appealing target for the malefactors. While fishing and scams can be overcome by educating people to adopt the best (and probably essential) cyber hygiene practices, other threats like viruses, Trojans, worms, ransomware, spyware, and adware may be impossible to identify for an average user. Therefore, cybersecurity specialists developed and continue improving malware detection methods. Malware detection consists of various techniques including signature-based, behavior-based, heuristic and machine learning-based detection. In particular, machine learning algorithms are considered to be pretty effective in identifying malicious behavior because they do not rely on pre-defined patterns and signatures of known malware, thus can identify previously unseen malware and adapt in future. ML (Machine Learning) techniques are often involved in other detection methods [16].

Machine learning-based malware detection requires the following activities: data collection and preprocessing, model training, testing and validation, deployment. Creating an extensive and suitable dataset is essential for any machine learning model. However, even if the model is trained on the best dataset at this moment of time, over time the data may and most likely will change. Feature relevance may vary significantly and even the data itself may have bias or errors, which will inevitably lead to model's predictive ability degradation. Such a process of ML model becoming less and less accurate in its predictions is called concept drift.

The phenomenon of concept drift has become a serious issue in Android malware detection [17] [18]. Constant measures taken by cyber security specialists are forcing cyber criminals to adapt and look for more inventive attack vectors and develop more sophisticated malicious software. In fact, sometimes even slight modifications in malware code make it undiscoverable for existing antiviruses.

Taking into account that the study of concept drift in Android malware is ongoing, it is possible to identify a wide range of research gaps. The identified gaps that will be covered in this thesis are the impact of using different feature sets (system calls and permissions), handling the bias towards majority class in the data by applying balancing techniques to the training datasets to improve model predictions. As concept drift causes degradation of ML-based malware detection solutions, the ML model must be periodically updated. Therefore, there is a need to compare techniques for updating an ML model: which techniques are the most convenient and allow to conquer concept drift in the best way. These updating strategies should provide answers on how much older data should be involved in retraining, and if incremental learning, which updates the model constantly with new data without the need to store old data, can provide acceptable performance comparable to full retraining. Lastly, the performance of the models should be evaluated accurately, which requires creating datasets which will reproduce concept drift conditions and will real-life data that malware detection solutions regularly face.

This thesis is an attempt to contribute to the study of concept drift in Android malware and has the following research questions:

- How do system calls and permissions perform as features in machine learning-based detection of evolving Android Malware?
- How do different balancing techniques impact the performance of the model when dealing with imbalanced training datasets in concept drift conditions?
- What are the most appropriate strategies for updating an ML model used for malware detection considering concept drift?
- How to simulate concept drift conditions in testing datasets to comprehensively evaluate ML-based mobile malware detection solutions?

This research proposes a novel approach of evaluation machine learning malware detection solutions by using two testing scenarios: testing on new data and on realistic data. This study also compares how non-stationary models perform using dynamic, static and hybrid features, while other works focus on dynamic features mostly (API calls). Another novelty point is extensive comparison of class weight, undersampling, oversampling and combined balancing techniques in non-stationary context.

2 Background Information and Related Work

This section is meant to familiarize the reader with the most important concepts, theories and technologies that are used in this thesis. It will also include a brief literature review that consists of studies on related topics, the findings from which were used in this research, as well as studies with similar experiment approaches.

2.1 Machine Learning

Machine learning is a way to make a computer accomplish tasks without being explicitly programmed. ML is applied when it is impossible to program strict rules and logic into software to solve a specific problem because this logic does not seem obvious, the rules might change over time or there are too many of them. Machine learning algorithms analyze data and find patterns in it to make predictions on new data. There are several approaches to machine learning such as supervised, unsupervised and reinforcement learning. In this thesis only supervised learning will be applied, which means that training data should be labeled.

Before machine learning models start predicting, they should be properly trained. Training stage requires a prepared dataset. Data preparation or preprocessing may include data cleaning, normalization, balancing, feature engineering, etc. Those procedures will be described below.

Another important step is selection of an appropriate ML algorithm. It is necessary to consider the problem type (e.g., classification, regression, clustering, etc.), dataset complexity and size, peculiarities of a specific algorithm. As the problem of malware detection is predicting a binary value, it requires a classification algorithm. The two algorithms used in this research are RandomForestClassifier and SGDClassifier provided by scikit-learn library [19].

2.1.1 RandomForestClassifier

Random Forests are very popular machine learning algorithms used in supervised learning for regression and classification problems. They are ensemble methods, meaning that they consist of several or many models and predict based on the individual predictions of these models. More specifically, random forests consist of ensemble of multiple decision trees. RF (Random Forest) algorithm randomly selects a subset of data and features, so called bootstrapping, and applies decision tree algorithms to each bootstrap. Each decision tree makes its predictions independently. While individual predictions might be highly inaccurate, the most frequent answer overall (in case of classification task) results in the much more precise final prediction [20]. Figure 2 shows an example of a RandomForestClassifier model that consists of multiple DecitionTreeClassifier models.

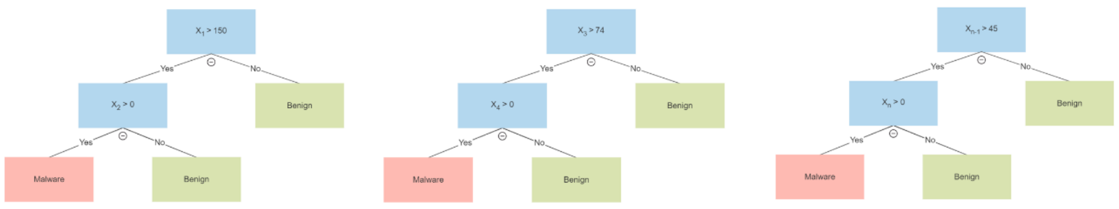


Figure 2. An example of a RandomForestClassifier model consisting of multiple DecisionTreeClassifiers. The features X_1, X_3 and X_{n-1} represent some system calls features, while X_2, X_4, X_n represent some permissions features.

RandomForestClassifier algorithm can be tuned by adjusting hyperparameters. For example, “n_estimators” is the number of trees in the model (default is 100), “max_depth” is the maximum depth of the tree (None by default). Most of the hyperparameters were left default, but “class_weight”, which is used to handle imbalanced dataset, and “warm_start” were relevant in this research.

Overall, random forest classifier was selected due to its high accuracy, resistance to overfitting (lack of generalization which results in high performance on the known data, but very poor on the unseen data) and ability to handle large datasets.

2.1.2 SGDClassifier

SGDClassifier is a linear algorithm that implements Stochastic Gradient Descent in order to optimize the model parameters. The algorithm works by iteratively updating the model parameters based on the gradient of the loss function with respect to the parameters. SGDClassifier is particularly useful for large datasets and high-dimensional data to overcome computational limitations [21]. However, the one particular feature that is important for this research is that SGDClassifier supports incremental learning. Incremental learning is a method of updating the model once new data becomes available. Full retraining can be annoying when data is constantly changing, or new data appears frequently. It also requires storing the previously available data, which may be impractical when the amount of data is very huge. Incremental learning in SGDClassifier is available via "partial_fit" method.

2.1.3 Normalization

In order to enhance the efficiency and stability of machine learning algorithms, normalization is a typical data preprocessing approach. It entails scaling the input features to a common scale. Normalization aims to ensure that each feature has a similar range and distribution to prevent the algorithm from favoring features with bigger scales [22]. It is important to mention that normalization can be necessary for some machine learning algorithms to work properly, while others may not need it at all. RandomForestClassifier and SGDClassifier handle normalization differently, which will be described in the Methodology section.

Normalization includes techniques like Standardization Scaling, Min-Max, Z-score normalizations, etc. In this research only MinMaxScaler and Normalizer methods from sklearn.preprocessing were used [23] [24]. MinMaxScaler scales features to a specified range, while Normalizer performs non-linear scaling (It works by scaling each row of the dataset to have a unit norm).

2.1.4 Balancing Techniques

In machine learning, the issue of imbalanced datasets, where one class or category of data is noticeably more or less represented than the others, is dealt with through balancing algorithms. When the dataset contains classes that are seriously imbalanced, the machine learning algorithms can bias towards the majority class, which would make it difficult to identify the minority class.

Balancing strategies include:

- Undersampling: the number of samples from the majority class is reduced to match the number of samples in the minority class.
- Oversampling: the number of samples in the minority class is increased to match the number of samples in the majority class.
- Combined: oversampling is followed by undersampling to clean up the noisy data generated by oversampling.
- Class weighting: higher weights are assigned to the minority class samples and lower weights to the majority class samples during the training process.

In this thesis six balancing techniques from imblearn library will be compared [25]. Undersampling methods are:

- RandomUnderSampler: fast and simple balancer that returns randomly selected subset of the majority class with the same size as minority class.
- NearMiss (version 3): the algorithm selects samples from the majority class that are farthest from the decision boundary.

Oversampling methods are:

- RandomOverSampler: generates new samples in the classes which are under-represented by duplicating the samples from minority class.
- SMOTE (Synthetic Minority Over-sampling Technique): creates new synthetic samples by linearly interpolating between the selected minority class sample and its k nearest neighbors.

Combined methods are:

- SMOTEENN: oversampling using SMOTE then applying ENN (Edited Nearest Neighbours) algorithm (the ENN algorithm works by identifying the majority class samples that are misclassified by their three nearest neighbors in the feature space and removing them from the training set).

- SMOTETomek: oversampling using SMOTE then applying Tomek Links algorithm (The Tomek Links algorithm works by identifying pairs of samples that are the closest to each other but belong to different classes and removes them) and repeats the procedure once more.

In addition to that, class weighting technique is included in the comparison. Class weighting can be an effective way to address the class imbalance problem, as it allows the model to give more importance to the minority class samples during training. This can lead to a better classification performance on imbalanced datasets, especially when the minority class samples contain important information for the task. Class weighting does not change the size of the dataset or the distribution of samples. However, assigning too high weight to the minority class samples may result in overfitting or model instability. Class weighting can be combined with other balancing techniques. In sklearn library when “class_weight” option is set to “balanced”, it calculates the weights so that they are inversely proportional to the frequency of samples in each class [26].

2.1.5 Evaluation Metrics

Evaluation metrics are used to assess the performance of a machine learning model. Evaluation metrics are different for classification and regression problems. The evaluation metrics for classification tasks used in this experiment are following:

- Accuracy: the proportion of correctly classified instances out of the total number of instances in the dataset. However, it is necessary to realize that when classes are unbalanced, accuracy can be deceptive. Accuracy is calculated like that:

$$Accuracy = \frac{Total\ number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{TN + TP}{TN + FN + TP + FP}$$

- Precision: the proportion of correctly classified positive instances (true positives) out of all positive predictions. Precision is useful in cases where there are high costs associated with false positives. Precision formula is:

$$Precision = \frac{TP}{TP + FP}$$

- Recall: the proportion of true positives out of all positive instances in the dataset. Recall is useful in cases where there are high costs associated with false negatives. Recall formula is:

$$Recall = \frac{TP}{TP + FN}$$

- F1 score: This is the harmonic mean of precision and recall. It is a balanced metric that takes both precision and recall into account. F1 score formula is:

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

2.2 Concept Drift

Concept drift is a phenomenon in machine learning and data science where the statistical properties of the target variable, or the relationship between the input and output data, changes over time. Concept drift usually leads to ML model degradation so that it becomes ineffective when applied to new data. Dealing with concept drift requires understanding the reasons that cause it in each situation and development of new techniques that will allow the model to adapt to novel data.

Concept drift is an especially relevant issue in the field of mobile malware detection. The nature and characteristics of mobile malware are constantly evolving, and new types of mobile malware are developed and distributed. That means statistical properties of the malware samples and their associated features will likely change over time, resulting in concept drift. For example, new versions of Android malware may use different obfuscation techniques or exploit other vulnerabilities than the previous versions. This can change the patterns and distribution of features that are important for malware detection, thus decreasing model's effectiveness. Additionally, new types of Android malware can appear that have not been seen before. It will probably result in noticeable concept drift in the ML-based malware detection models. Statical properties and behavior patterns can differ between new malware types and the malware present in a training dataset, causing the model either producing false positives or, vice-versa, failing to detect new malware samples.

2.3 Related Work

This section provides a review of the relevant literature and research that has been conducted in the field of study of mobile malware, concept drift in malware detection, used to contribute to the development of this thesis.

First of all, the dataset used in this research is Kronodroid Dataset [27]. Its development is described in this paper [28]. The work identifies the problem of Android malware evolution being understudied and manages to generate a dataset that allows to study concept drift in Android Malware. The resulting dataset is labeled, diverse, timestamped, covering 2008-2020 years of Android history, including a set of static and dynamic features [28].

There is another relevant article [29]. Its objectives include research of usage of system calls as features in an Android concept drift-related study and the mitigation strategies of the impact of concept drift on ML model that detects android malware even in case of imbalanced datasets [29]. In the experiment, periodized Kronodroid datasets are used based on "last modification" and "first seen" timestamps. The research managed to build an effective model and minimized retraining and provided importance of system calls features. The study allows further research on permissions as features for ML-based malware detection, balancing techniques and model updating strategies.

Another paper studies the sustainability of "yearly dataset-based trained classifier" using a combined feature set of API calls and permissions [30]. It follows two approaches: non-incremental learning (when model is trained on a specific year dataset and tested on all 8 test datasets separately) and incremental learning (when new train datasets are introduced iteratively to the model). It is important to indicate that incrementally trained Random Forest is performed via setting “warm_start” hyperparameter to true. However, this approach cannot be characterised as incremental learning, which is explained in Appendix 2.

An article “Are Machine Learning Models for Malware Detection Ready for Prime Time?” questions the nearly perfect performance of the models published in scientific papers because they fail to reproduce it in real-life conditions [31]. The article raises the problems of relative sizes of malware and benign classes, malicious and benign data time periods not matching and improper training when the model receives knowledge about future data. This issues are also covered in details in another study which introduces the idea of spatial bias and temporal bias and justifies the use of 10% portion of malicious samples in the test dataset to reproduce the real-world data [32].

Another study evaluates the impact of datasets on machine learning-based Android malware detection [33]. Some of its conclusions state that class imbalance in training datasets influences the performance of ML model seriously and that features can evolve over time so that the further the timeline of malware datasets are, the more significantly feature importance will change.

An incremental batch-learning malware detection solution was proposed in this study [34]. Its approach is to update the model once concept drift was detected using classification error rate, then update the model incrementally using sequential deep learning. However, this research also indicates a catastrophic forgetting problem of incremental learning approach.

Lastly, there is a study that uses a transfer-based learning approach to update model with new data [35]. The clear advantage of such a solution is that the model preserves its knowledge of historical data avoiding bias towards the newly arrived data.

3 Methodology

This section describes methods and procedures used to conduct the research. It covers dataset selection, dataset transformation to simulate concept drift conditions, data normalization and balancing, machine learning models, algorithm of testing the model on new data and evaluation.

This thesis follows a quantitative research approach, specifically it shows an experiment where ML learning model is tested using a dataset consisting of malicious and benign Android applications. Such an approach will provide empirical prediction capabilities of an ML model in different scenarios. The machine learning algorithm will be used in supervised binary classification problem because the target (y) will be either malware (1) or benign (0).

3.1 Data

Any machine learning algorithm depends on the dataset because it provides the foundation for both the algorithm's training and evaluation. The performance of the ML method can be significantly impacted by the dataset's size, quality, and diversity.

A dataset that is used to study concept drift phenomenon should meet several requirements. First of all, the dataset should consist of time-stamped data, so it can be divided into smaller datasets consisting of data from a certain time period. Next, the data should cover a significant period (in case of Android malware it means the period covering a great portion of Android malware history) allowing concept drift to emerge. At the same moment, the data must be more or less equally distributed over the time period in order to avoid creating empty or deficient subsets. The dataset should be also meeting the requirements such as multiple data sources, sufficient size, accurate labelling, etc.

3.1.1 Dataset selection

The dataset selected for this thesis is the real device dataset from the Kronodroid Project as it was specifically created to explore concept drift and meets all requirements for the dataset selection mentioned above [28] [27].

The most important characteristics of the Kronodroid real device dataset according to the KronoDroid paper [28]:

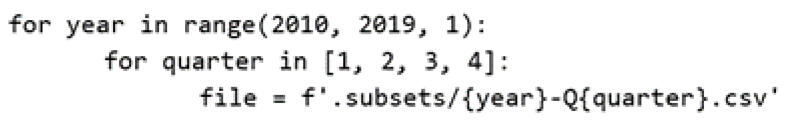
- Contains 41,382 malware (from 240 malware families) and 36,755 benign samples; 78,137 samples in total.
- Time frame is 2008-2020.
- Includes 289 dynamic and 200 static features.
- Available in .csv format.

The features needed for the experiment are system calls (collected during dynamic analysis phase on real device with ARM architecture), permissions (extracted during static analysis phase), timestamps and the label. System calls are represented by 288 numeric features, where value is an absolute frequency of an exact system call produced by a running application. Permissions are represented by 166 standard permissions that values are binary (1 – permission is requested, 0 – permission is not requested). Timestamps are available in 4 ways: Earliest Modification, Last Modification, First Seen VT, First Seen in the Wild (ITW). The label is a binary feature where 1 denoted malware and 0 indicates benign application [28].

3.1.2 Dataset transformation

There are multiple machine learning approaches that can be used to study concept drift. For example, the data stream learning approach allows updating the model continuously with the newly arrived data, while in a more traditional batch learning approach a fixed dataset is fitted into the model at once. In this research batch learning is preferred because in case of malware detection solutions the essential part for updating the model is creating a new training dataset. New training datasets must consist of new malware and benign samples that are appropriately labelled which is time consuming. Updating the ML model every quarter of the year seems to be a realistic task for that.

Therefore, the first step to make with the Kronodroid dataset is to split it into the time-based subsets (batches). The timestamp feature is chosen to be Earliest Modification. Time period is a quarter of a year: Q1 – months 01-03 (January - March), Q2 – months 04-06 (April - June), Q3 – months 07-09 (July – September), Q4 – months 10-12 (October – December). The dataset covers 2008-2020 years, however in this research it was decided to limit the timeframe by 2010-2018 range due to the better malware-benign data distribution. While separating the data into the corresponding periodized subsets, data cleaning is performed and the features that will not be used are dropped. The remaining features are all numeric types. Consequently, 36 datasets were created. The naming formatting is “<yyyy>-Q<quarter>.csv”, for example, “2011-Q2.csv” or “2017-Q3.csv”. Such naming allows to read the subsets in iterative manner in the program in the following manner (see Figure 3):



```
for year in range(2010, 2019, 1):
      for quarter in [1, 2, 3, 4]:
            file = f'.subsets/{year}-Q{quarter}.csv'
```

Figure 3. Example of Python code to iterate through the subsets.

After achieving 36 datasets where each one consists of data corresponding to a specific time frame, it is important to recreate real life conditions. There are two important aspects in real life data: 1) it never consists of new data only but rather includes a generous amount of older data; 2) the ratio between malicious and benign data should not be random but rather well-justified [31]. The malware-goodware distribution is shown in Figure 4.

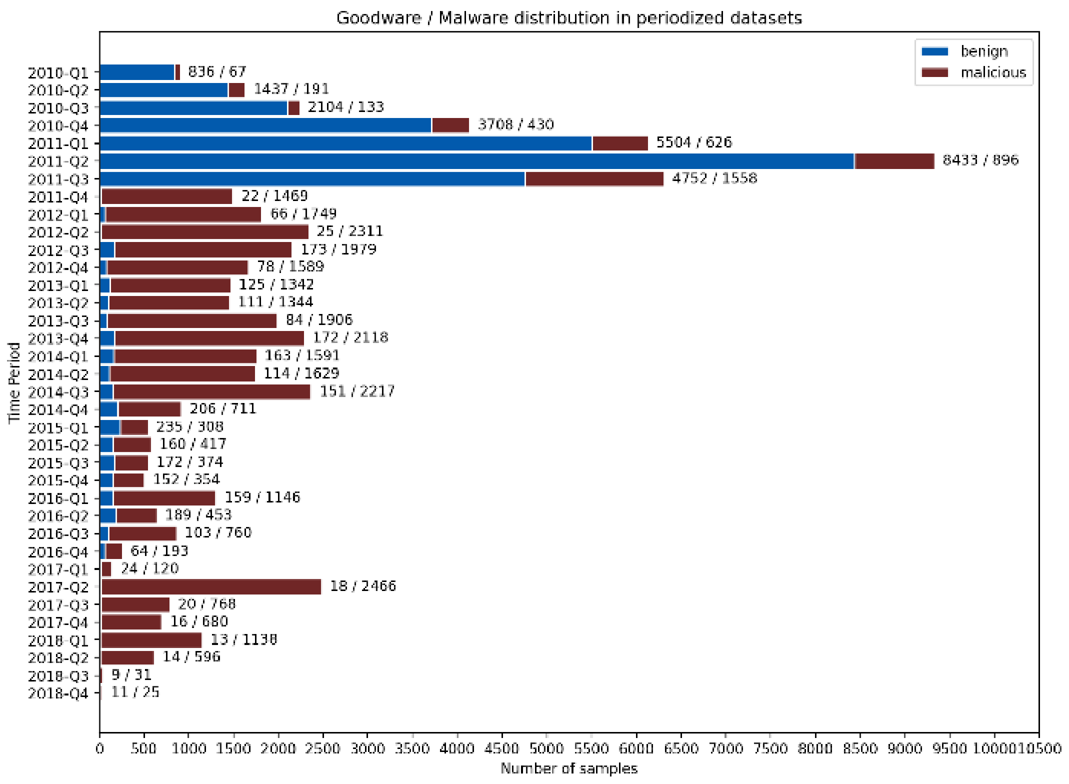


Figure 4. Distribution of benign and malicious data in the periodized datasets.

The first aspect can be solved by adding some percentage of the data from the previous datasets to the current one. In this experiment, it is assumed that every dataset will include 20% of data from each previous dataset (it is rough approximation, in reality the portion of older data will likely vary drastically). However, the same data should not be used in training and testing. In order fix this issue, it is necessary to create separate datasets for training and testing. 2010-Q1 is divided 80% to 20%, 80% is used to train the model, while 20% is kept. 2010-Q2 is divided the same way, 80% of 2010-Q2 dataset with addition of the 20% of 2010-Q1 are used to test the model trained on the 80% of the 2010-Q1. At the same time 80% of the 2010-Q1 can be concatenated with 80% of 2010-Q2 to be used as training dataset which will be tested on 80% of 2010-Q3 plus kept 20% of 2010-Q2 and 20% of 2010-Q1, in case of model retraining. This process is shown in Figure 5. In this case the distribution of goodware and malware in test datasets be seen in Figure 6.

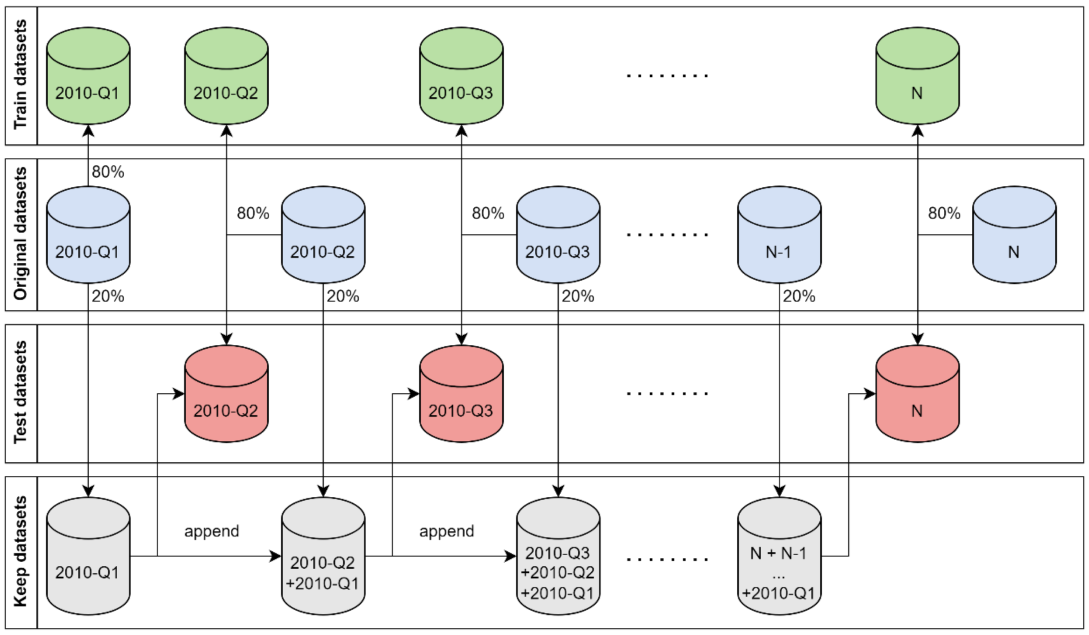


Figure 5. Algorithm used for creating realistic datasets. Original periodized datasets are presented in blue colour, train datasets – green, test datasets – red, keep datasets (storing 20% of each previous dataset which is never used for training) – grey.

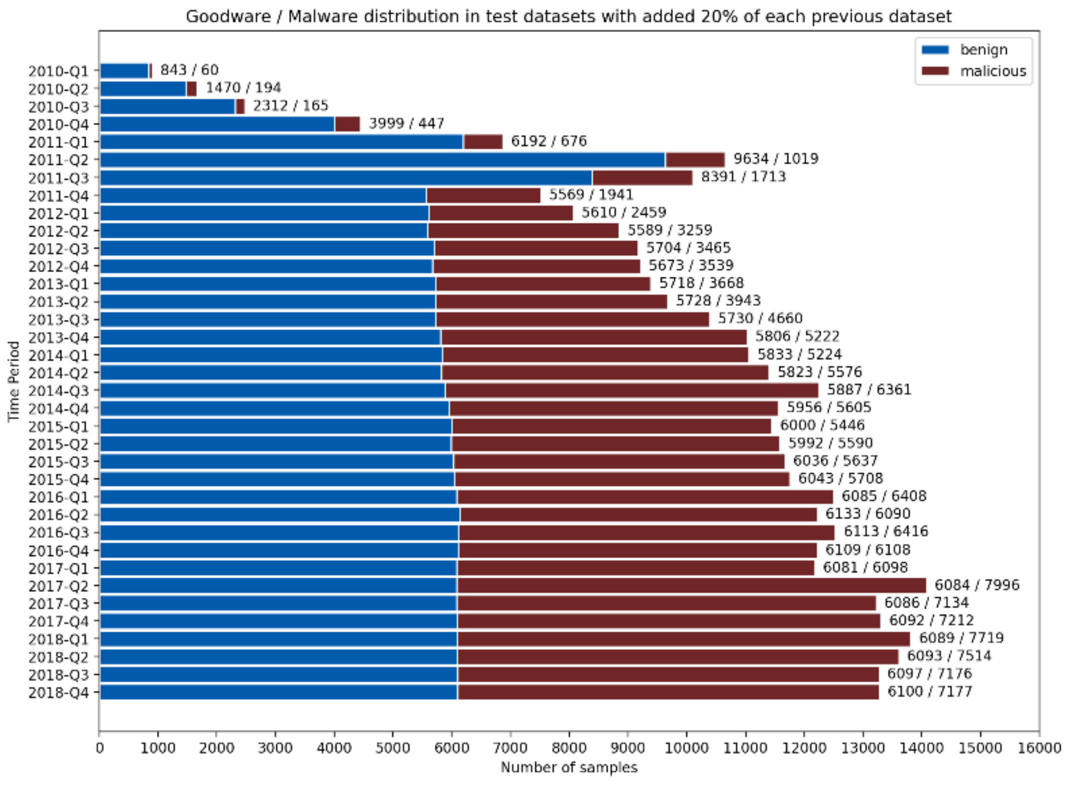


Figure 6. Distribution of benign and malicious data in the test dataset with addition of 20% of data from each previous dataset.

The answers for the second aspect can be found in “TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time” article [32]. The research questions the inflated results of some ML classification models due to the spatial and temporal biases. According to the paper, spatial bias is caused by unrealistic assumptions about the ratio of benign and malicious samples in the data, while temporal bias refers to inaccurate time splits of training and testing sets so that future data reserved for testing, to be included during the training phase. The research comes to the conclusion that in reality the malware portion in all the applications is between 6% and 18.8% and decided to use 10% malware in its datasets for convenience [32]. In this research the test dataset where malware portion is not 6-18%, the samples of the class which exceeds its range (18% for malware or 94% for benign) are undersampled, forming realistic datasets (also referred as “real_data”). The distribution of goodware and malware in these test datasets can be seen in Figure 7.

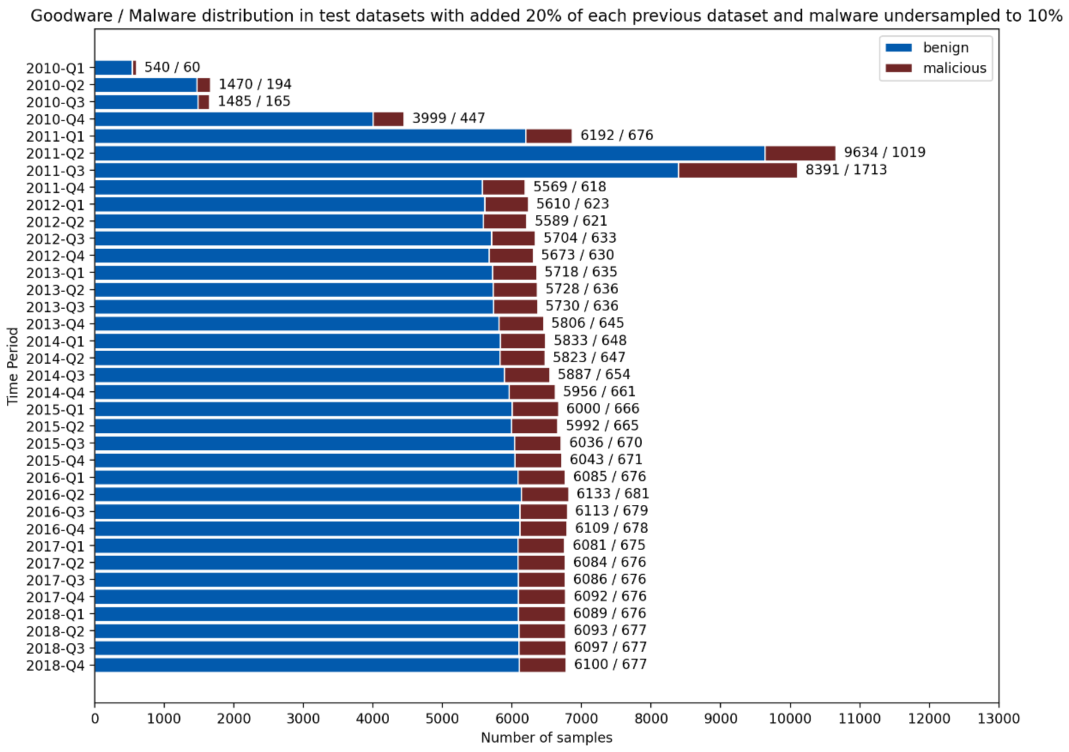


Figure 7. Distribution of benign and malicious data in realistic test datasets.

As a result, there are two different testing scenarios: 1) using datasets that consist only of data that corresponds to a specific time frame (also referred as “new_data”); 2) using realistic dataset that include older samples and have approximately 9:1 goodware-malware ratio. Comparing the performance of the ML models tested in both scenarios should provide a better understanding of Android malware concept drift.

3.1.3 Data normalization

While permissions in the dataset are represented by the binary values (1 – permission is requested, 0 – permission is not requested), system calls are numeric values that depict absolute frequencies of each system call. Top 10 system calls by the greatest max value are 'read': 529603, 'gettid': 348961, 'clock_gettime': 338569, 'gettimeofday': 333057, 'futex': 284834, 'write': 247397, 'pread': 226826, 'getuid32': 224928, 'SYS_310': 191120, 'SYS_329': 167813. Thus, it may be useful to scale system calls values, meaning that the value should be in the range of 0 to 1. The scikit-learn module provides MinMaxScaler() and Normalizer() functions to perform data normalization. However, different machine learning algorithms handle normalized data with various levels of success. In Appendix 3 there are two figures that show performance of ML model in case when MinMaxScaler(), Normalizer() and no normalization are applied. In Figure 17, the ML algorithm is RandomForestClassifier, and the results show that no normalization showed the best performance, followed by Normalizer() and MinMaxScaler() being the worst. At the same time, in Figure 18 where the chosen algorithm is SGDClassifier, Normalizer function outperformed the two others.

3.1.4 Data balancing

Examining balancing techniques in conditions of concept drift is one of the objectives of this thesis. As can be seen in Figure 4, the datasets are imbalanced: the ratio between malicious and benign samples is far from 1:1. It will cause the model trained on those datasets to be biased towards the more represented class. In order to avoid this bias, there are two possible solutions: 1) use resampling techniques; 2) adjusting class weights.

The Imbalanced-learn python library provides various methods to cope with the issue of imbalanced data [25]. In particular, imblearn includes oversampling (such as RandomOverSampler, SMOTE), undersampling (such as RandomUnderSampler, CondensedNearestNeighbour, NearMiss) and combined techniques (such as SMOTEENN, SMOTETomek).

At the same time, scikit-learn classification algorithms support “class_weight” parameter [19]. By default, all classes have equal weight, however there is an option “balanced” that adjusts weights inversely proportional to the class frequencies, and an option to manually set the class weights.

The figures comparing how the model performs with different balancing techniques and without any balancing will be provided in the Results section. The assessment of the impact of balancing techniques will be provided in the Discussion section.

3.2 Machine Learning Models

Selection of a proper machine learning model is a pivotal step in creating a successful predictive model. As the data label is binary (either malicious or benign), the machine learning algorithm has to be a classifier. The first choice was RandomForestClassifier from scikit-learn library [36]. The random forest classifier is a powerful algorithm that works well for many different classification problems, especially those that require dealing with complex data. The algorithm gained popularity because of its high accuracy, effectiveness dealing with the noise and overfitting, and simplicity of usage. However, during the research it revealed that RF classifier does not support incremental learning, despite the fact that in some studies parameter “warm_start” is set to “True” is pretended to be incremental learning [30]. Therefore, the need for an ML algorithm that supports incremental training appeared. There are 5 classification algorithms suited for incremental training in scikit-learn library: naive_bayes.MultinomialNB, naive_bayes.BernoulliNB, linear_model.Perceptron, linear_model.SGDClassifier, linear_model.PassiveAggressiveClassifier. Empirically selected algorithm was SGDClassifier from sklearn.linear_model [37]. Using “partial_fit” allows to train the model incrementally without the need of full retraining.

In terms of hyperparameter tuning, parameter “n_jobs” is set to “-1”, which speeds up the computational time by using all available CPU cores. Other parameters are default if not specified exactly.

3.3 Experiment scenario

In order to generate concept drift conditions, the following algorithm was developed. Machine learning model is trained on the first dataset is tested on the data from the second dataset, then model is updated with consideration of the new data (fully retrained or incrementally trained). In such an iterative manner the model is being evaluated on 35 datasets (from 2010-Q2 to 2018-Q4) and updated with respect to those datasets. In case of full retrain, the datasets are concatenated and all together are fitted to the model, whereas in case of incremental learning only the new dataset is passed to the model using “partial_fit” method. In addition, there will be a review of the performance of models trained on last N subsets (for example, last_3, last_5, etc.).

3.4 Evaluation and visualization

Any meaningful conclusions can be drawn about the performance of an ML model only according to the properly selected evaluation metrics. In this study, the following metrics will be provided: accuracy, precision, recall and F1-score for each time-period. However, it is important to indicate that recall is the most important metric for malware detection problems. Recall value depends on the number of False Negatives, as a result, it reflects model's ability to correctly identify all positive samples. Detecting all malicious samples is a priority for the ML model.

An extremely important part for understanding the performance of the model is visualization of the result. In this thesis visualization is performed using Matplotlib library for python [38]. The typical results figure will consist of 4 plots (accuracy, F1-score, recall and precision) which compare metrics of several models.

3.5 Project availability

The full project is available on GitHub [39].

4 Results

In this section the different comparisons will be shown on the graphs, which will allow to assess these results and draw the conclusions.

4.1 System calls vs Permissions

The comparison of system calls, permission and both combined is performed using RandomForestClassifier model which is fully retrained every quarter. Figure 8 and Figure 9 show the results when model is tested on new data and realistic data respectively. Figure 1 summarizes the results and shows averages of each metric.

Table 1. Comparison of system calls and permissions

Features	Performance (%)							
	New data				Realistic data			
	Accuracy	F1	Recall	Precision	Accuracy	F1	Recall	Precision
System calls	84.5	84.9	78.4	94.9	95.3	78.2	82.9	77.9
Permissions	84.6	84.6	78.5	93.8	96.1	81.8	88.0	77.9
Both	90.8	90.1	85.4	97.3	97.7	88.3	89.1	89.3

Tested on new data, system calls and permissions show almost identical results in average (see Table 1). However, Figure 8 shows that permissions had more significant drops in accuracy and recall throughout the experiment, like in 2015-Q4, 2016-Q4, 2018-Q3 and 2018-Q4. Figure 8 also shows that using both feature sets model's results fluctuate, however the fluctuation range of permissions is greater. System calls also perform slightly better in terms of precision. Overall, system calls marginally outperform permissions when tested on new data.

At the same time, when the models are tested realistic datasets, permissions start to outperform system calls (see Table 1 and Figure 9). For example, permissions have

5.1% higher average recall than system calls. Figure 9 also shows that there are very few fluctuations performance when testing on realistic datasets, but rather general trends (decreasing, increasing, etc.). For example, all feature sets show significant increase in recall up to 2013-Q1, then reaching plateau, while in precision system calls decline gradually until 2015-Q1 and then remain constant, but permission initially drop in 2011-Q2, then rise again in 2011-Q3, followed by gradual decent up to 2014-Q3, then remaining stable. Since 2014-Q3 permissions show greater results in precision than system calls by 5-8%. Considering that permission also outmatched system calls in recall during almost all periods, permissions can be declared as a more reliable set of features for detecting android malware in realistic conditions. In both cases (when tested on new data or realistic data) the combination of system calls and permissions noticeably improves the predictions of the models.

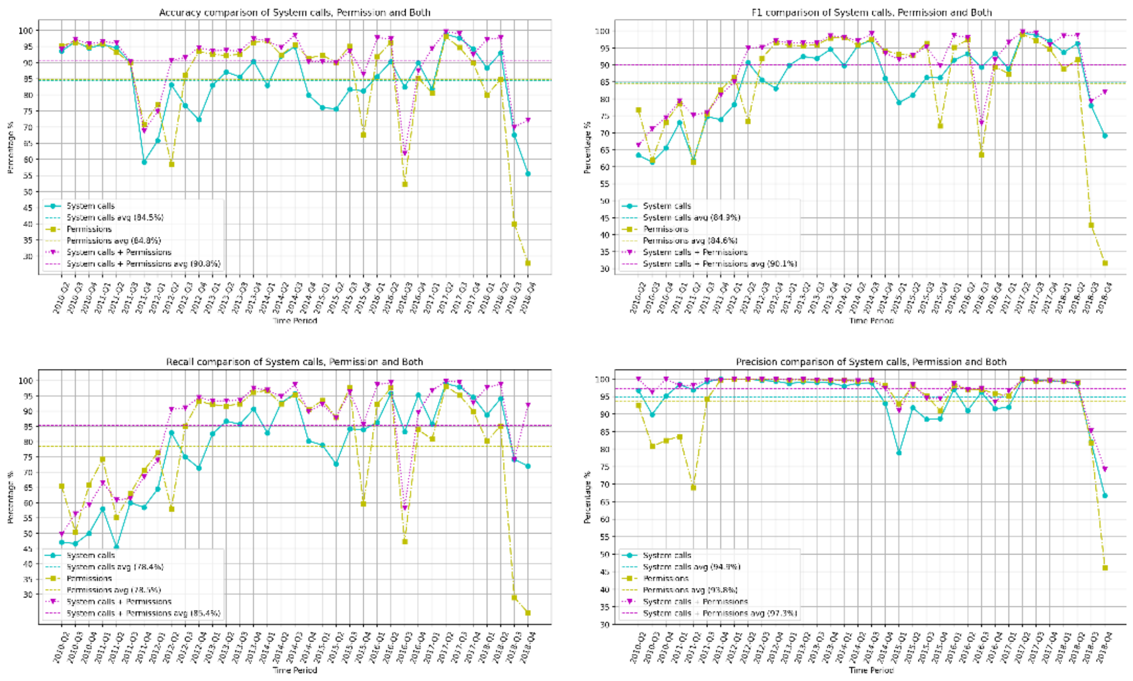


Figure 8. Evaluation metrics comparison of RandomForestClassifier model where selected features are system calls, permissions or both tested on new_data.

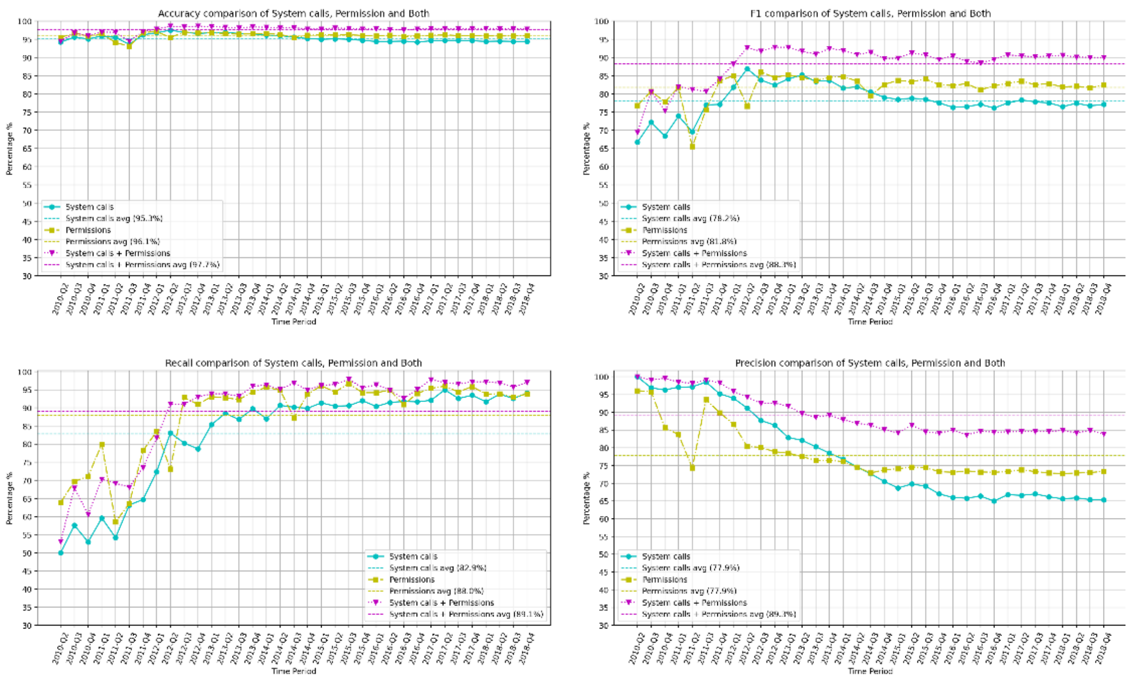


Figure 9. Evaluation metrics comparison of RandomForestClassifier model where selected features are system calls, permissions or both tested on real_data.

4.2 Balancing

The balancing techniques were represented by three categories: undersampling, oversampling and combined. Undersampling includes RandomUnderSampler, NearMiss, oversampling – RandomOverSampler, SMOTE, combined – SMOTEENN, SMOTETomek. The option when RandomForestClassifier hyperparameter "class_weight" is set to "balanced" is tested as well. The results when none balancing is applied are presented for convenience.

Table 2 shows the average results of balancing, as well as CPU execution time to assess the computational complexity of a balancer. The top three balancers by average recall are RandomUnderSampler, NearMiss, SMOTE. Their performance can be seen in Figure 10 when tested on new data and in Figure 11 when tested on realistic data.

Table 2. Comparison of balancing techniques

Category	Balancer	Average performance (%)								CPU time (min)
		New data				Realistic data				
		accuracy	F1	recall	precision	accuracy	F1	recall	precision	
Undersampling	RandomUnderSampler	92.0	90.7	90.6	92.3	96.9	86.0	93.3	80.5	10.6
	NearMiss	92.2	90.6	90.4	92.2	96.0	83.3	92.9	76.8	35.0
Oversampling	RandomOverSampler	91.1	90.6	86.5	96.7	97.7	88.4	89.8	88.6	11.8
	SMOTE	91.6	91.1	88.2	95.3	97.5	88.0	91.6	85.6	35.9
Combined	SMOTEENN	87.7	87.4	83.8	92.7	93.2	84.2	85.1	83.3	198
	SMOTETomek	91.5	91.1	88.0	95.3	95.5	88.2	89.8	86.7	188
Class_weight=”balanced”		90.4	89.5	84.7	97.1	97.6	87.9	88.5	89.3	12.2
None		90.6	90.1	85.4	97.2	97.7	88.3	89.0	89.4	9.5

Figure 10 shows almost identical results that RandomUnderSampler, NearMiss, SMOTE balancers have starting from 2012-Q2 in accuracy and recall and from 2011-Q4 in precision. From 2010-Q3 to 2011-Q3 there is a massive drop in precision, especially for undersampling techniques RandomUnderSampler and NearMiss, while SMOTE’s drop was much less significant.

Figure 11 shows similar picture for accuracy and recall in case of testing on realistic data. However, the precision results differ more significantly. SMOTE performed the best in terms of precision, outclassing undersamplers during almost all time periods and never dropping below 80%, while RandomUnderSampler and NearMiss had drops up to 52% and 35% respectively.

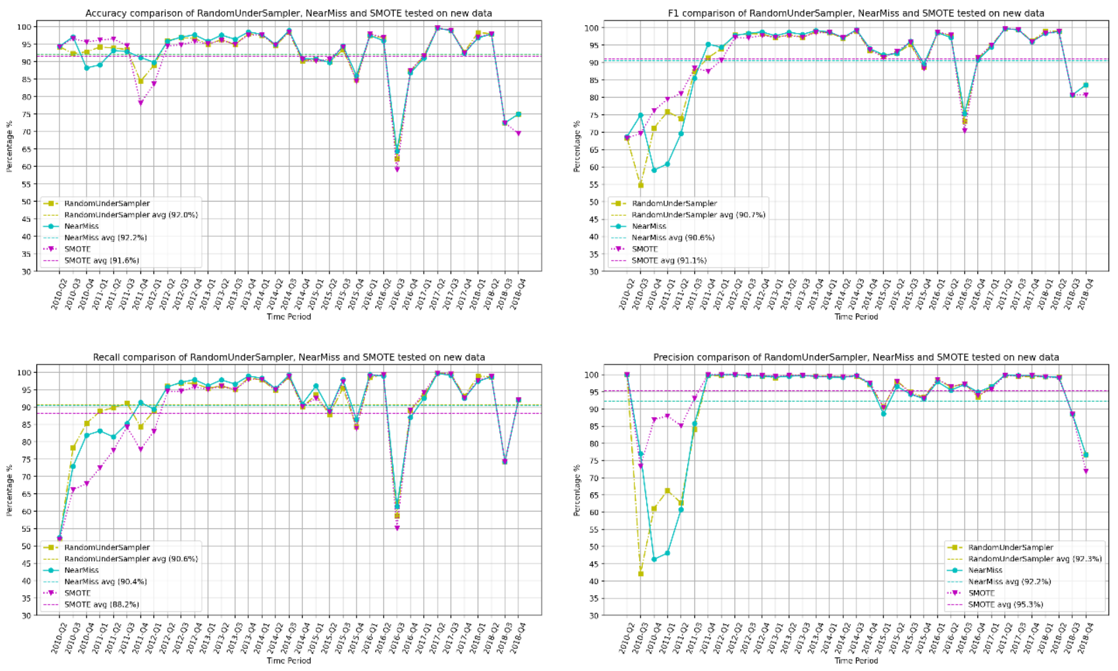


Figure 10. Evaluation metrics comparison of top three balancing techniques tested on new data.

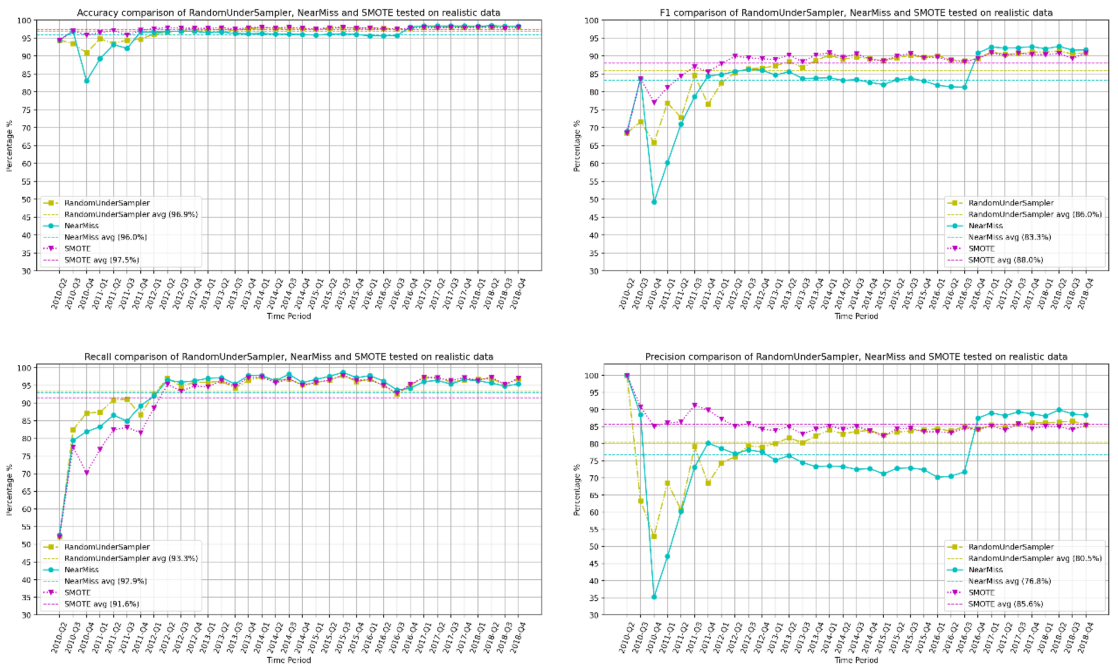


Figure 11. Evaluation metrics comparison of top three balancing techniques tested on realistic data.

4.3 Model Updating

Model updating is performed every quarter of the year, however, the updating of the model can be done in several different ways. This experiment compares full retrain, incremental training and retraining on last N datasets.

Figure 12 and Figure 13 show the performance of RandomForestClassifier models retrained on the last 1, last 3, last 5 datasets and full retraining tested on new data and realistic data respectively.

Figure 12 shows that the difference between model retrained on last 1, 3 and 5 datasets and full retraining is very slight when tested on new data. All metrics graphs show similar patterns. Generally, the model updated with dataset consisting of the greatest amount of most recent data performs the best. In terms of accuracy and precision the results are last 1 > last 3 > last 5 > full retrain, although the difference is marginal. In precision the results are just the opposite, however the average scores vary in the range of 1%, which is almost negligible.

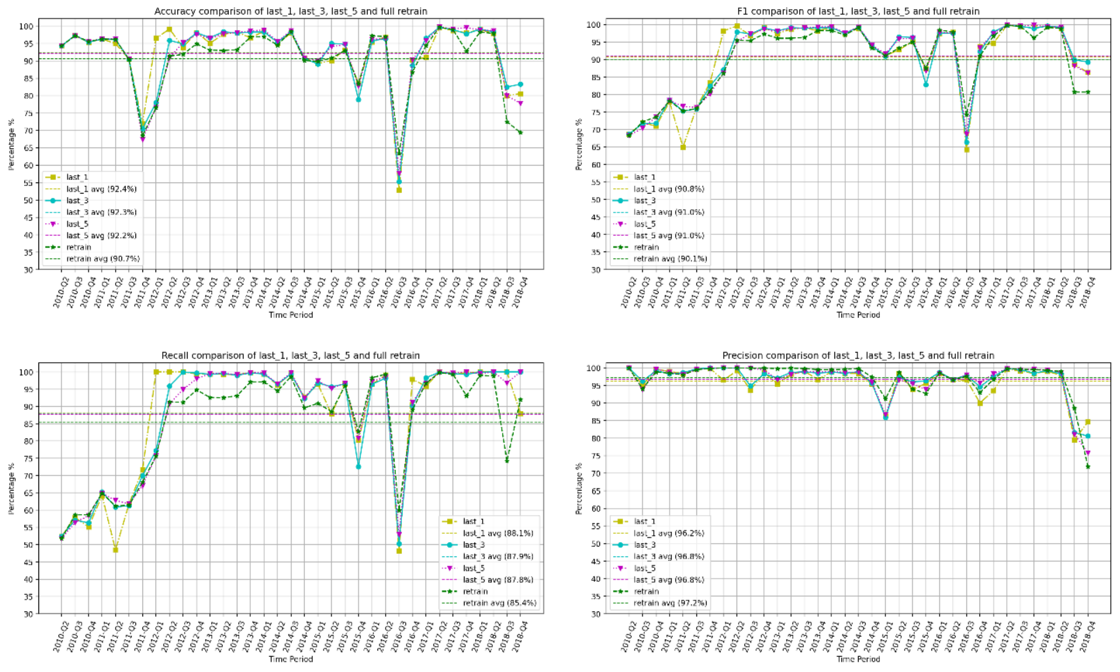


Figure 12. Evaluation metrics comparison of RandomForestClassifier models retrained on last 1, 3, 5 and all datasets tested on new_data.

On the other hand, Figure 13 presents a very different picture when the model is tested on realistic data. In this case, full retraining is the only approach showing adequate performance. Obviously, it is due to the fact that realistic data includes a lot of older samples from the previous datasets, which requires the model to accumulate knowledge compared to testing scenario with new data only.

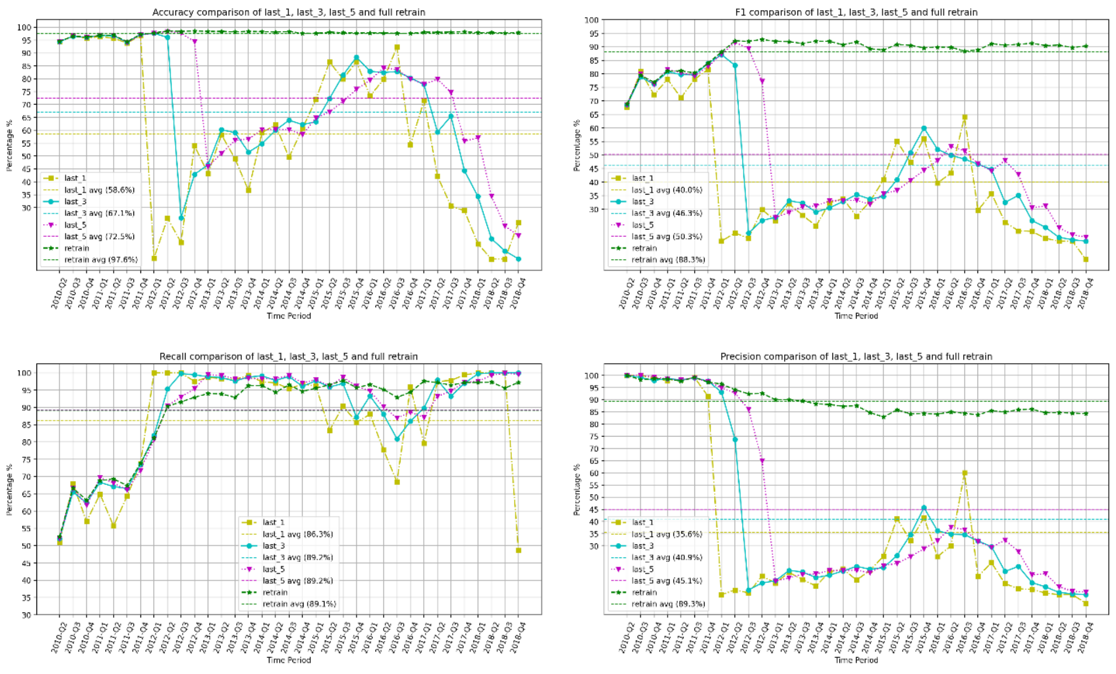


Figure 13. Evaluation metrics comparison of RandomForestClassifier models retrained on last 1, 3, 5 and all datasets tested on real_data.

Taking into account that RandomForestClassifier does not support incremental learning, this approach should be evaluated by implementing another ML algorithm. Therefore, SGDClassifier is introduced. Figure 14 and Figure 15 present the results of SGDClassifier models that are retrained on last 1 dataset, incrementally trained and fully retrained tested on new data and realistic data respectively.

Figure 14 shows that incremental model slightly outperforms retrain on last 1 dataset, and more noticeably outperforms full retraining option in accuracy and recall when tested on new data.

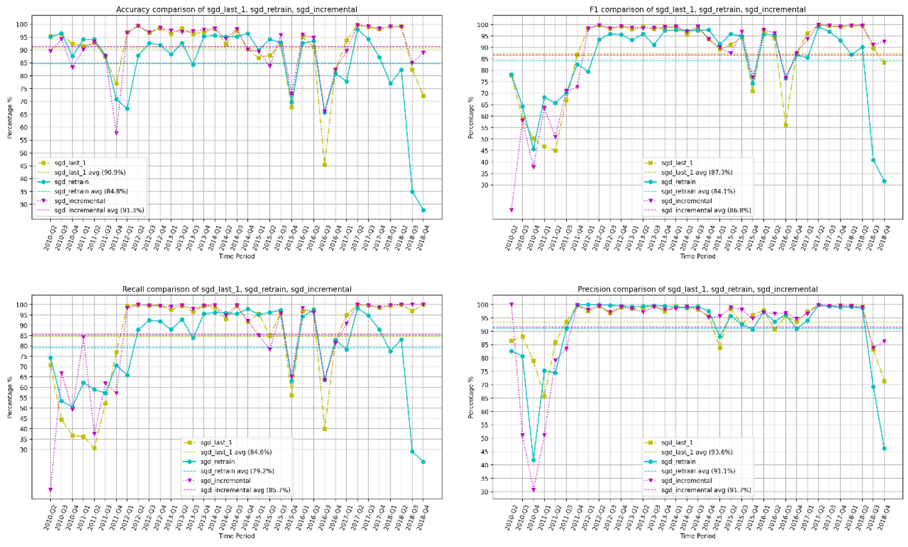


Figure 14. Evaluation metrics comparison of SGDClassifier models retrained on last_1, all datasets and incrementally trained (tested on new_data).

Just like with random forest model, Figure 15 shows how fully retrained SGDClassifier model dramatically outclassed the competitors when tested on realistic data. Its accuracy remained stable around 95%, recall gradually grew until 2013-Q4, while precision was slightly decreasing over time. In comparison, incremental model and last_1 model graphs follow the same trends of constant fluctuations and huge drops, meaning incremental model has substantial bias towards new data and tendency to forget about older data that was previously trained on. Nevertheless, incrementally trained model has considerably better average accuracy (by 3.8%) and recall (by 3%).

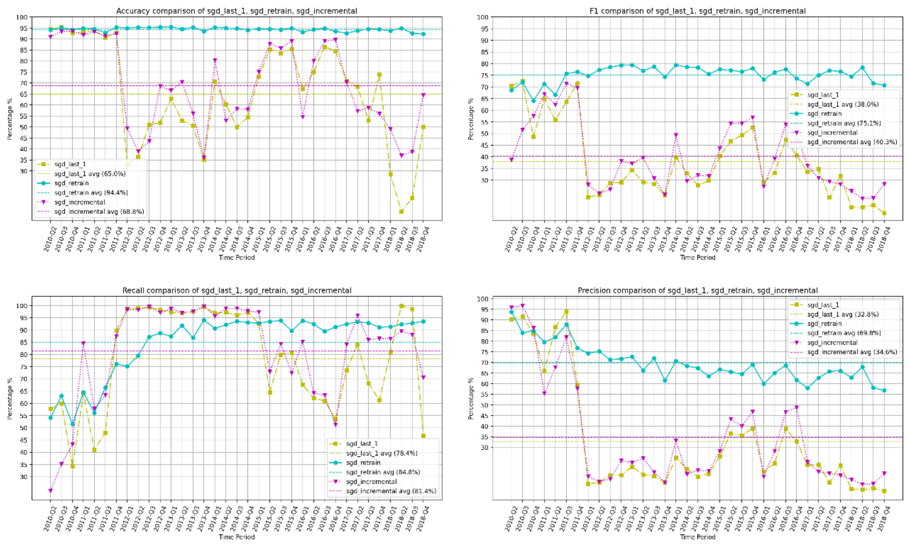


Figure 15. Evaluation metrics comparison of SGDClassifier models retrained on last_1, all datasets and incrementally trained (tested on real_data).

5 Discussion

This section provides the analysis of the result, describes limitation and opportunities for the further research.

5.1 General observations

Recall is used as a key evaluation metric because it indicates how well a model is able to identify true positive cases. When recall value is low it means that many malicious samples are labelled as benign, despite the fact that overall accuracy is high enough. In the malware detection problem, it is desperately important to find as many malware samples as possible. Precision, which can measure false positives, is secondary in this scenario. The interesting trend is that precision decreases as recall rises, demonstrating so called precision-recall tradeoff. Recall is increased by reducing the number of false negatives, which can lead to an increase of false positives – decreasing the precision.

Another observation is that on all recall graphs (less often on precision and accuracy) the initial few periods show unpleasant results. It is probably caused by the distribution of malware and benign data in corresponding training datasets. Figure 4 shows that until 2011-Q4 benign class seriously prevails over malicious.

5.2 Comparison of system calls and permissions

System calls and permissions are both actively used in malware detection. System calls are the ways a program interacts with the operating system, when permissions are a set of rules that determine which actions a program is allowed to perform in the operating system.

Despite similar results system calls and permissions showed on the new data, permissions very noticeably better when tested on realistic data. As a possible explanation of this result, although the action that malware performs in the operating systems can vary over time, the permissions, it requests to obtain it desirable goal, may stay the same. It is also necessary to indicate that both permissions and system calls complement each other and using both as feature will result in much better performance of the model.

5.3 Comparison of balancing techniques

In comparison of balancing techniques, the best tree balancers by recall value came out to be RandomForestClassifier, NearMiss and SMOTE in both scenarios, when tested on new and realistic data. All the balancers except SMOTEENN had similar scores in recall and outperformed no balancing option (see Table 2). Interestingly, the best results in recall were shown by undersampling methods. However, undersampling reduces the number of samples in the majority class. That may potentially lead to the loss of valuable information. Precision graphs of Figure 10 and Figure 11 show that SMOTE oversampling technique helped to avoid dramatic drops in the range from 2010-Q to 2011-Q2. Counterintuitively, option “class_weight=balanced” performed poorer than no balancing. The purpose of it may be the fact that in training datasets malware is not always the minority class, thus, providing benign samples more weight might increase noise of the data and worsen the performance of the model.

Another interesting thing to compare is CPU Execution time. It describes how much computational power the balancer requires. This may be an important parameter when retraining the model frequently on large imbalanced datasets, although it might not be a problem when the model is updated every quarter of a year as there are enough resources and time for retraining. RandomUnderSampler and RandomOverSampler ran about 10% longer than without any balancing, running SMOTE and NearMiss took approximately 3 times longer, SMOTEENN and SMOTETomek took whopping amount of time – six times longer than SMOTE.

Overall, oversampling imbalanced training datasets using SMOTE is the most reliable solution.

5.4 Comparison of updating strategies

In comparison of different updating strategies testing on new data and realistic data show the opposite results. When the models were tested on the new data, the performance decreased as more older data was included in training a model. Model retrained on the last dataset has the highest results (see Figure 12). It means that when the task is to recognize completely new unseen malware, the older data is unimportant and only the most up to date datasets should be used in training. However, when the test dataset is realistic, meaning that it also includes some amount of older data, the results are reversed (see Figure 13). In this scenario only full retrain option show sufficient performance, when the results of retraining on the last 1, 3 and 5 are inadequate starting from 2012-Q1, especially in precision and accuracy.

In order to compare another promising model updating strategy – incremental learning, an alternative ML algorithm – SGD Classifier was selected, because it supports incremental learning using “partial_fit”. Interestingly, incremental learning approach outperformed not only full retrain, but last_1 option as well when tested on new data (see Figure 14). Although full retrain was still a clear winner when tested on realistic data, the incremental training was still noticeably better than last_1 (see Figure 15). Taking into consideration that incremental learning was performed by using new training data passed into the “partial_fit”, the incrementally trained model has a bias towards a newer data. The possible improvements that can be made are described in the Limitations and further work section.

5.5 Testing datasets

The experiment conducted in this research proved the need for different testing scenarios. To evaluate how model can handle unseen data, it can be tested just with new data. However, this approach does not provide understanding of how the model will behave in the real world. To simulate the realistic testing datasets with concept drift for comprehensive evaluation of machine learning-based malware detection solutions, the two steps are proposed. Firstly, include older data in the testing datasets. In this research each period specific testing dataset includes 20% of data from each previous dataset. Secondly, the portion of malicious samples in the testing datasets should correspond to approximately 10%. It can be achieved by undesampling the malicious data or oversampling the benign data, considering overall number of samples to avoid creating very small datasets.

5.6 Limitations and further work

This thesis is an attempt to explore some aspects of a wide and understudied topic – concept drift in malware detection. Therefore, it is necessary to indicate the limitations of this work as well as propose possible directions for further research.

Firstly, the period for updating the research is selected to be a quarter of the year for convenience. It is possible to experiment with this period, making it shorter or longer. Secondly, the amount of older data in the realistic datasets in this experiment is selected to be 20% of each previous dataset. It is a rough assumption, because in reality this value can differ crucially. It may be interesting to study how much older data an antivirus faces in real-life and recreate this percentage in an experiment.

In this research only ML-based malware detection approach was considered. It can be valuable to combine several malware detection approaches and test them in concept drift conditions. For example, using signature-based approach to detect known malware and ML-based to detect unseen. Splitting the roles can improve the detection rate.

Lastly, incremental training approach seems to be the most suitable approach in malware detection problem for unseen samples. Although, in this experiment it still loses to full retrain option when tested on realistic dataset, there are some potential improvements that can change it. For example, appending some amount of older training data to the current can increase the performance of incrementally trained model towards the older data [34].

6 Conclusion

Malware changes and adapts over time to overcome the existing security measures. The evolution of malware is the main reason why the current detection solutions degrade over time, which often is referred to as concept drift. This thesis contributes to the studies of concept drift in the field of malware detection.

A dataset of Android Malware was selected and transformed into the period specific subsets. The period is quarter of the year. Two separate test datasets were created, consisting of new data (to evaluate how models handle unseen data) and realistic data (to simulate the real-world condition).

Two machine learning algorithms were used for binary classification problem of malware detection: RandomForestClassifier and SGDClassifier. The research managed to answer all the research questions.

System calls and permissions were compared as feature sets for ML-based Android malware detection method. The research concludes that permissions might be more sustainable in concept drift conditions. Next, various balancing techniques were evaluated, with SMOTE being the most reliable option. Also, the comparison of updating strategies concluded that when handling the new data only it is only necessary to train the model on the newest available training data. Full retraining remains the clear winner when tested on realistic data. However, incremental learning has the potential to become the most convenient approach for updating models in the real-world environment if the problem of forgetting older data is solved. Lastly, the research proposed an algorithm of creating testing datasets imitating real-world data.

References

[1] N. Islam and R. Want, “Smartphones: Past, Present, and Future,” *IEEE Pervasive Comput.*, vol. 13, no. 4, pp. 89–92, Oct. 2014, doi: 10.1109/MPRV.2014.74.

[2] “Before IPhone and Android Came Simon, the First Smartphone - Businessweek.” https://web.archive.org/web/20120701034025/http://www.businessweek.com/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone (accessed Apr. 12, 2023).

[3] “Apple Reinvents the Phone with iPhone,” *Apple Newsroom*. https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/ (accessed Apr. 12, 2023).

[4] “Apple Reports Fourth Quarter Results,” *Apple Newsroom*. https://www.apple.com/newsroom/2007/10/22Apple-Reports-Fourth-Quarter-Results/ (accessed Apr. 12, 2023).

[5] “Android Open Source Project.” https://source.android.com/ (accessed Apr. 12, 2023).

[6] “Mobile Operating System Market Share Worldwide,” *StatCounter Global Stats*. https://gs.statcounter.com/os-market-share/mobile/worldwide/ (accessed Apr. 12, 2023).

[7] A. Cranz, “There are over 3 billion active Android devices,” *The Verge*, May 18, 2021. https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021 (accessed Apr. 12, 2023).

[8] “Smartphones vs Cameras: Closing the gap on image quality,” *DXOMARK*, Mar. 19, 2020. https://www.dxomark.com/smartphones-vs-cameras-closing-the-gap-on-image-quality/ (accessed Apr. 12, 2023).

[9] “Desktop vs Mobile Market Share Worldwide,” *StatCounter Global Stats*. https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/ (accessed Apr. 12, 2023).

[10] A.-T.-T. I. I.-S. Institute, “AV-ATLAS - Malware & PUA,” *AV-ATLAS - Malware & PUA*. https://portal.av-atlas.org/malware/statistics (accessed May 10, 2023).

[11] “Mobile cyberthreat report for 2022,” Feb. 27, 2023. https://securelist.com/mobile-threat-report-2022/108844/ (accessed May 11, 2023).

[12] M. Labs, “Goldoson: Privacy-invasive and Clicker Android Adware found in popular apps in South Korea,” *McAfee Blog*, Apr. 12, 2023. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/goldoson-privacy-invasive-and-clicker-android-adware-found-in-popular-apps-in-south-korea/ (accessed May 11, 2023).

[13] “Play Protect,” *Google Developers*. https://developers.google.com/android/play-protect (accessed May 11, 2023).

[14]“Android Version Market Share Worldwide,” *StatCounter Global Stats*. https://gs.statcounter.com/os-version-market-share/android (accessed May 11, 2023).

[15]“iOS Version Market Share Worldwide,” *StatCounter Global Stats*. https://gs.statcounter.com/ios-version-market-share/ (accessed May 11, 2023).

[16]Ö. A. Aslan and R. Samet, “A Comprehensive Review on Malware Detection Approaches,” *IEEE Access*, vol. 8, pp. 6249–6271, 2020, doi: 10.1109/ACCESS.2019.2963724.

[17]D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, “The Concept Drift Problem in Android Malware Detection and Its Solution,” *Secur. Commun. Netw.*, vol. 2017, pp. 1–13, 2017, doi: 10.1155/2017/4956386.

[18]A. Guerra-Manzanares, S. Nõmm, and H. Bahsi, “In-depth Feature Selection and Ranking for Automated Detection of Mobile Malware:,” in *Proceedings of the 5th International Conference on Information Systems Security and Privacy*, Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2019, pp. 274–283. doi: 10.5220/0007349602740283.

[19]F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Mach. Learn. PYTHON*.

[20]L. Breiman, “Random Forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[21]L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization Methods for Large-Scale Machine Learning.” arXiv, Feb. 08, 2018. Accessed: May 07, 2023. [Online]. Available: http://arxiv.org/abs/1606.04838

[22]D. Singh and B. Singh, “Investigating the impact of data normalization on classification performance,” *Appl. Soft Comput.*, vol. 97, p. 105524, Dec. 2020, doi: 10.1016/j.asoc.2019.105524.

[23]“sklearn.preprocessing.MinMaxScaler,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (accessed May 07, 2023).

[24]“sklearn.preprocessing.Normalizer,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.preprocessing.Normalizer.html (accessed May 07, 2023).

[25]G. Lemaıtre and F. Nogueira, “Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning”.

[26]“sklearn.utils.class_weight.compute_class_weight,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html (accessed May 07, 2023).

[27]aleguma, “About the Kronodroid Dataset.” Apr. 11, 2023. Accessed: Apr. 13, 2023. [Online]. Available: https://github.com/aleguma/kronodroid

[28] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, “KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization,” *Comput. Secur.*, vol. 110, p. 102399, Nov. 2021, doi: 10.1016/j.cose.2021.102399.

[29] A. Guerra-Manzanares, M. Luckner, and H. Bahsi, “Android malware concept drift using system calls: Detection, characterization and challenges,” *Expert Syst. Appl.*, vol. 206, p. 117200, Nov. 2022, doi: 10.1016/j.eswa.2022.117200.

[30] W. Cho, H. Lee, S. Han, Y. Hwang, and S. Cho, “Sustainability of Machine Learning-based Android Malware Detection Using API calls and Permissions,” in *2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, Laguna Hills, CA, USA: IEEE, Sep. 2022, pp. 18–25. doi: 10.1109/AIKE55402.2022.00009.

[31] L. Cavallaro, J. Kinder, F. Pendlebury, and F. Pierazzi, “Are Machine Learning Models for Malware Detection Ready for Prime Time?,” *IEEE Secur. Priv.*, vol. 21, no. 2, pp. 53–56, Mar. 2023, doi: 10.1109/MSEC.2023.3236543.

[32] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, “TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time”.

[33] X. Ge, Y. Huang, Z. Hui, X. Wang, and X. Cao, “Impact of datasets on machine learning based methods in Android malware detection: an empirical study,” in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, Dec. 2021, pp. 81–92. doi: 10.1109/QRS54544.2021.00019.

[34] A. A. Darem, F. A. Ghaleb, A. A. Al-Hashmi, J. H. Abawajy, S. M. Alanazi, and A. Y. Al-Rezami, “An Adaptive Behavioral-Based Incremental Batch Learning Malware Variants Detection Model Using Concept Drift Detection and Sequential Deep Learning,” *IEEE Access*, vol. 9, pp. 97180–97196, 2021, doi: 10.1109/ACCESS.2021.3093366.

[35] Y. Sun, K. Tang, Z. Zhu, and X. Yao, “Concept Drift Adaptation by Exploiting Historical Knowledge,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4822–4832, Oct. 2018, doi: 10.1109/TNNLS.2017.2775225.

[36] “sklearn.ensemble.RandomForestClassifier,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed Apr. 17, 2023).

[37] “sklearn.linear_model.SGDClassifier,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (accessed Apr. 17, 2023).

[38] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, 2007, doi: 10.1109/MCSE.2007.55.

[39] dmmart, “dmmart/AndroidMalwareConceptDrift.” May 11, 2023. Accessed: May 11, 2023. [Online]. Available: https://github.com/dmmart/AndroidMalwareConceptDrift

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – warm_start for incremental learning

Figure 16 shows how poorly RF model performs with "warm_start" hyperparameter enabled when trying to learn the model incrementally. According to sklearn Glossary of Common Terms and API Elements, "warm_start" uses the previously fitted model's parameters to initialize a new fit. If new batches of data are large enough and are sufficiently different (which is to be expected in concept drift conditions), they might overwrite the model, meaning it will forget about the data it was trained previously. "partial_fit" is behaves differently. It should be used for incremental training as it allows to update the model with new data, keeping knowledge earned training on previous data. However, RandomForestClassifier does not support "partial_fit", which forced me to use another algorithm that supports this option.

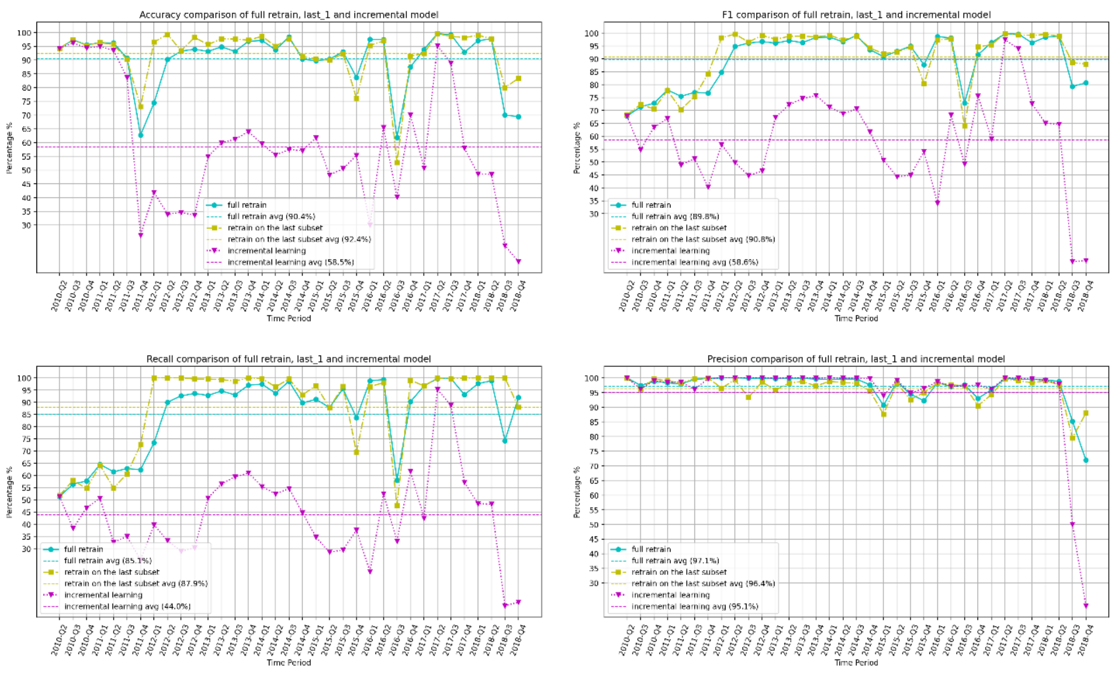


Figure 16. RandomForestClassifier model incremental learning through enabling "warm_start" hyperparameter compared to full retraining and retraining on last dataset tested on new_data.

Appendix 3 – Data normalization impact on RF and SGD models

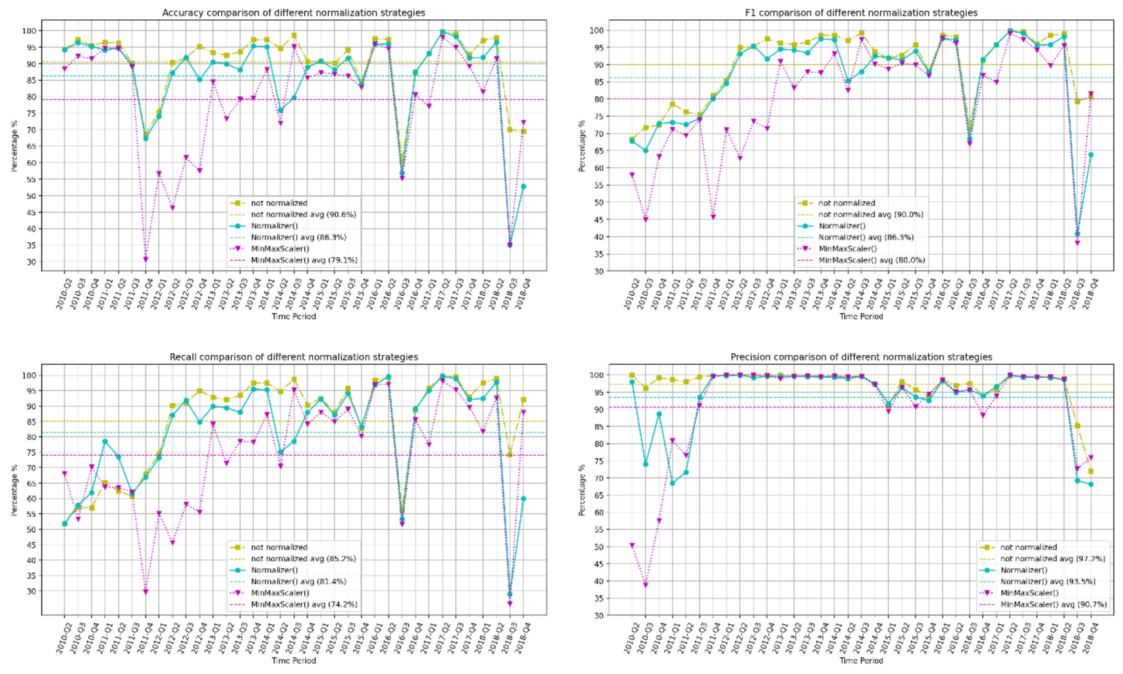


Figure 17. Normalization of data impact on RandomForestClassifier model.

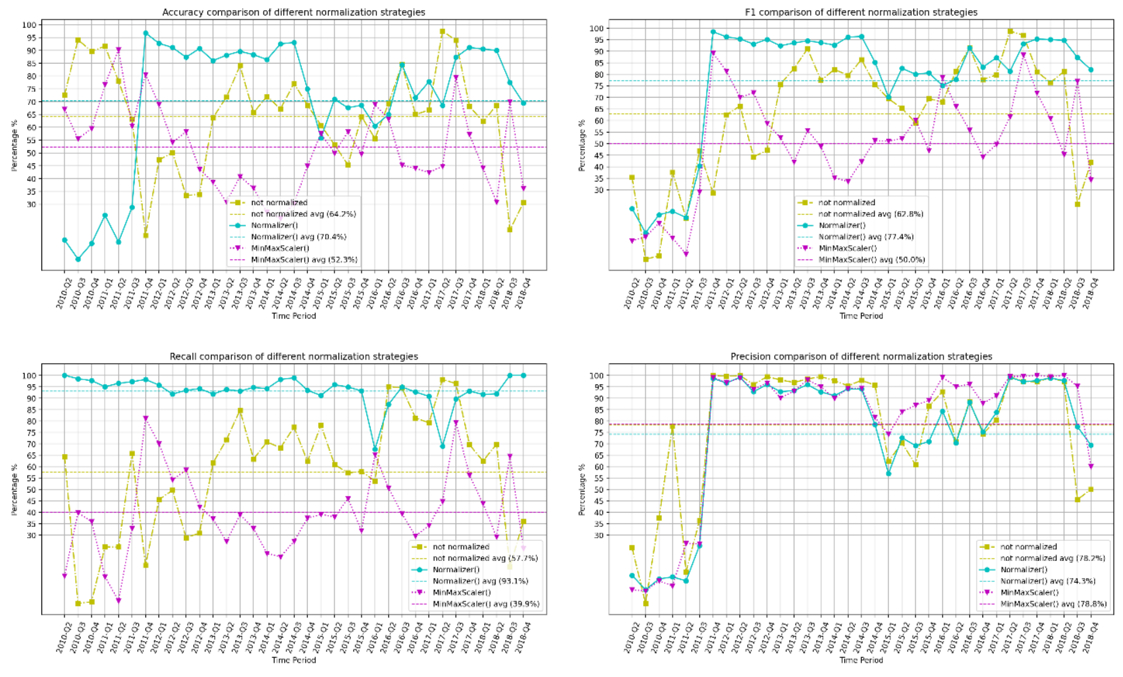


Figure 18. Normalization of data impact on SGDClassifier model.