Android Malware category classification

EIRIK GRANVIN BAKKE, Norwegian University of Science and Technology, Norway

The Android operating system has become the most popular mobile phone operating system, therefor the number of malicious application targeting Android devices have increased significantly. Machine Learning (ML) have become a highly used tool for malware prevention and detection. In this study we have created ML models that classifies an Android application as benign or a specific malware category. There are two ML algorithms that was used, XGBoost and multi-layered perceptron (MLP). Both algorithm attained good recall and precision scores for the multiclass problem. One XGBoost model attained had the best performance with a 0,98 and 0,95 for recall and precision respectively, and a low number of false for the benign class.

CCS Concepts: • **Computing methodologies** → **Neural networks**; *Supervised learning by classification*; **Boosting**.

Additional Key Words and Phrases: XGBoost, MLP, Malware, Android malware, Malware family, malware classification

ACM Reference Format:
Eirik Granvin Bakke. 2023. Android Malware category classification. 1, 1 (December 2023), 12 pages.

1 INTRODUCTION

Android is the leading Operating system for mobile devices, with over 70% of all mobile phones running Android OS [5]. Being the most used mobile device operating system with over three billion active annual users, Android have become the number one target for mobile device malware [4, 17].

There have been a significant growth in Android Malware over the past 10 years. Figure 1 show the significant increase in android malware from 2013 to 2023. The figure show that in 2013 there were just over 1 million Android malware, and in 2023 the number has increased to over 34 million [1].

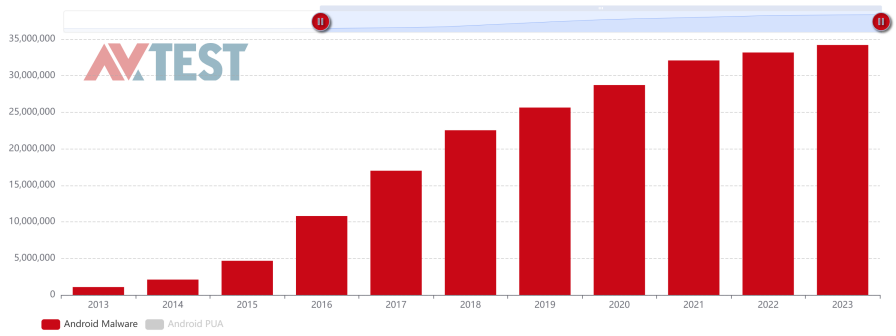

Fig. 1. Android malware statistics from 2013 until 2023 [1]

Author's address: Eirik Granvin Bakke, eirikgba@stud.ntnu.no, Norwegian University of Science and Technology, Gjøvik, Norway.

Manuscript submitted to ACM

Machine learning is one approach to prevent the use of malicious applications. ML can prevent downloads of malicious applications and detect malicious applications that exist on a Android device. Multi-layered perceptron (MLP) and XGBoost (extreme Gradient Boosting) are examples of two algorithms that have showed good performance for classification problem in general, but also specifically for malware classification [2, 12, 13, 15]. XGBoost have shown good performances in machine learning competitions since it was developed in the middle of 2010s, including winning and being in the top 10 at multiple competitions [3, 13]. The competitions include classification problem for malware data. Multiple research has presented good classification metrics by MLP model for malware classification problems. The studies [2, 12, 15] findings shows that MLP models can have high accuracy scores and be fast classifiers for malware ML problems.

The objective for this study is to create and compare XGBoost and MLP for a multiclass classification of Android malware. The paper will go into each of the proposed model individually and compare with each other. The dataset used for the study will contain malicious sample divided into malware categories and a benign class. A notable part will be to look into how well the model separate malware from benign, including classifying the malware in the correct malware category.

The rest of the paper is organized as follows, section 2 goes into some background information on relevant material and existing research on the topic. The Methodology 3 talks about the different neural network models and how the classification problem is approached. 4 Results dives into the result of the respective models, and lastly 6 concludes the paper.

2 RELATED WORK/BACKGROUND

Classification has become a central part in security of information and communication technology system and devices. Establishing something as malicious or not is an important task of many security mechanisms. There exist many machine learning (ML) algorithms that can be used for classifications problems. This research has the task of classifying an Android application as benign or as a specific malware category type. For this problem XGBoost and MLP are proposed algorithms that are used.

XGBoost is a algorithm that was created by Tianqi Chen and Carlos Guestrin in 2014. It is a boosting algorithm and more specifically based on the gradient tree boosting technique (also known as gradient boosting machine (GBM). The code for the algorithm is available as an open source package [1]. Being a boosting algorithm, the model are build of multiple weaker models, with the aim of combining to a high performing model. The model will go through each of the weak models sequentially, where the next model will try to correct the result from the previous model. The XGBoost model combines multiple weak tree algorithms to create a single model [3].

XGBoost algorithm have advantages in addition to having high performances, XGBoost is highly scalable. According to Chen and Guestrin the XGBoost model can be ten times faster than other popular models as of the time their paper was released. The speed of their algorithm was a result of multiple different techniques. Chen and Guestrin highlighted XGBoost use of out-of-core computation as the most important technique that resulted in the high performance [3].

The XGBoost models have been some of the best performing models in machine learning competitions in the end of the 2010 decade. Both Nielsen, and Chen and Guestrin refers to XGBoost being used most out of the winning and best preforming machine learning algorithms in different competitions. According to Chen and Guestrin XGBoost was used in 17 out of the 29 published solutions for Kaggle competitions in 2015. They also mentioned that all teams in the top

[1] https://github.com/dmlc/xgboost/

ten of the KDDcup 2015 used XGBoost [3]. In his Master thesis Nielsen brought up some reasons why XGBoost models have done so well in competitions. Firstly he mentioned that additive tree models in general can handle highly complex relationships given the model have a sufficient number of trees with sufficient depth. The most essential reason was the models being adaptive and flexibility. Being adaptive makes so the models can fit to multiple and highly complex data. The course of dimensionality problem in machine learning is handle well by the XGBoost algorithm, because it automatically performs feature selection as a result of is adaptive feature [3, 13].

Multi layer perceptron

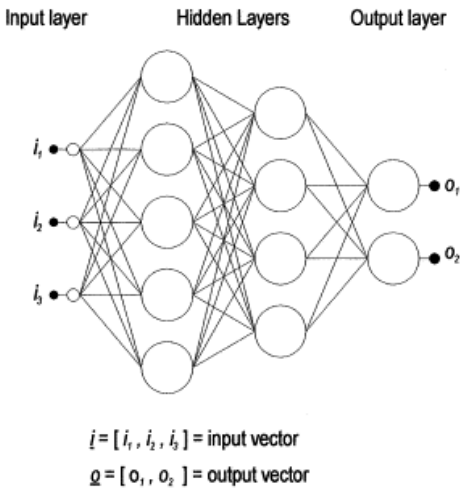

Fig. 2. Example of MLP model with two hidden layers, three input nodes and two output nodes [7]

Multi-layer perceptron (MLP) is one type of artificial neural networks. A MLP model has input and output layer and one or more hidden layer between them, and each layer contains some nodes. The output layer has a minimum of one node, and for classification problem the number of nodes is determine by the classification problem. Classification using MLP models handle both binary and multi class data. For binary classification the model has one or two nodes, but one node is the most common. For multi class classification problem, the number of nodes is decided by the number of classes, where each node represent one class. The number of nodes in the input layer is based on the number of features in the dataset, where each node represent one feature. A MLP model have one or more hidden layers, with one or more hidden nodes. Figure 2 [7] shows an example on how an MLP model can be structured, with two hidden layers. Multi-layered perceptrons are highly interconnected. Each node in a layer is connected to every node in the subsequent layer. [7, 9, 14]

Activation function Activation functions are important for neural networks computations. Without an activation functions the neural network could not handle complex data relationship, and would work as a Linear Regression Model [18]. Sigmoid is one example of an non-linear activation function, and it is one of the most used activation functions. In a neural network each node has an activation foundation related to it, that calculate the output of the layer, either as input to the next layer or as the output of the model. A single neural network can use multiple activation functions a different places in the network. The activation function in a model can have a significant impact on the quality of the model [6, 18]. Under are four activation's function used in this research, and the mathematical equation for each:

Sigmoid [9, 18]

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Tanh (hyperbolic tangent function): [6, 18]

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Softmax [10, 18]

$$f_j(z) = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}, \tag{1}$$

ReLU [6, 18]

$$Relu(z) = max(0, z)$$

3 METHODOLOGY

The objective of this study was to analyze and compare two machine learning models', XGBoost and MLP, performance on a multiclass classification problem. The model's goal is to classify a sample as a specific malware category or benign. The CCCS-CIC-AndMal2020 dataset was used and the models were implemented using Python.

3.1 Dataset

The dataset used is the CCCS-CIC-AndMal2020 dataset, which consist of 400k android apps. The dataset was created in collaboration by the Canadian Center for Cyber Security (CCCS) [2] and the University of New Brunswick based the Canadian Institute for Cybersecurity (CIC) [8], and is made publicly available[3]. Their are a total of 200k benign and 200k malware samples in the dataset. The benign samples are collected from the Androzoo dataset, which gets it samples from multiple official android markets. The malware samples are collected from CCCS data, and VirusTotal [4] was used to categorize the malware. The dataset have a total of 14 malware categories: Adware, Backdoor, FileInfector, No Category, Potentially Unwanted Apps (PUA), Ransomware, Riskware, Scareware, Trojan, Trojan-Banker, Trojan-Dropper, Trojan-SMS, Trojan-Spy and Zero Day. Each category consist of multiple malware families with similar characteristics, e.g. adware category contains the likes dowgin, inoco and zdtad families [8, 16]. The dataset is distributed for both static and dynamic analysis. For this study the static analysis dataset is used.

Both Keys et al. and Rahali et al. shows in their figure 3 the dataset is highly unbalanced [8, 16]. The Riskware class has almost half of all malicious samples, with over 97k instances, and has less than one thousand fewer number of instances than the combination of the other malicious samples. The number of Trojan-Banker instances is 887 and number of File Infector instances is 669 in the dataset, meaning the Riskware class is over one hundred times the size of each of the two smallest classes [8, 16].

Train & test data The dataset was split into data for training and test dataset to asses the performance of the model. The division of the dataset was distributed with the test data was 25% of the complete dataset while the training dataset was the largest with 75% of the complete dataset. Sklearn library was used to divided the data into randomized subsets, using the Train Test Split function.

[2] https://www.cyber.gc.ca/en
[3] Dataset is available at: https://www.unb.ca/cic/datasets/andmal2020.html
[4] https://www.virustotal.com/gui/home/upload

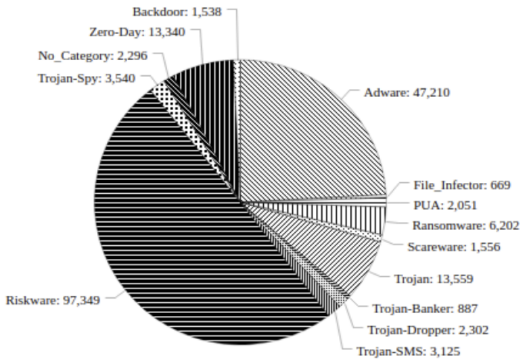

Fig. 3. Malware category distribution in Dataset [8, 16]

3.2 Incremental Feature Selection

To reduce the number of features for the machine learning models feature selection was implemented. For this study feature selection was essential, because of the size of the dataset. The feature selection step improved the interpretability, and reduced the training time and computational resources needed by the ML models.

The large size of the dataset combined with the computational resource limitation when preforming this study, incremental feature selection was implemented. Liu and Setiono [11] proposed a method for using incremental probabilistic algorithms for feature selection on a large dataset. Their approach was using the Las Vegas Filter (LVF) algorithm for feature selection on chunks of the data at a time. In their study the feature selection algorithm was implemented on multiple dataset. The concept was an algorithm, named LVI, that has three inputs, the dataset, allowed inconsistency rate and a percentage of the data used. The algorithm created a subset of the dataset and used the LVF algorithm on the subset, and returned a set of features that met the inconsistency rate criteria [11].

For this study incremental feature selection was implemented using the K-Best model, on a chunk of the data. The model took in a specified size of the dataset, and implemented the feature selection model on this subset. The model was configured to find the two best features from each subset and append them to a list with the selected features. Duplicates of the features in the list was removed.

3.3 Implementing XGBoost

The XGBoost algorithm is open source, as mentioned in the previous chapter. The implementation of the XGBoost open source library was implemented in python. The XGBoost-Classifier was imported from the XGBoost library and used in the python program. The XGBoost algorithm can take a set of parameters that specifies the configuration of the model. The number of estimators, meaning the number of boosting rounds or trees the model that is used. The maximum depth of trees is another parameter used. The depth is calculated as the path from the root node to the leaf node, where each node in the path is one step. With a maximum depth of three, e.g. the tree will have four or less nodes in each path from the root to the leaf including both nodes.

The XGBoost algorithm's attribute of having the ability to perform feature selection, and the quality of the feature selection was also tested. Two XGBoost model was created, one on the full dataset and one on the dataset with feature

Table 1. Hyperparameter optimization

Hyperparameter	Values
Evaluation metric	'rmse', 'mae', 'merror', 'mlogloss', 'auc'
Objective/function	'multi:softmax', 'multi: softprob'
Number of estimator	50, 100, 200,
Max depth of trees	50, 100, 200,
Learning rate	0.01, 0.1, 0.3

selection. Both model was initially created with the same parameters, softmax function, multiclass log-loss as evaluation metric, max depth of the tree is three and 100 boosting rounds.

Hyperparameter optimization was implemented of the model with feature selection. Hyperparameter optimization is the process of tuning a machine learning model to find parameters that makes the ML perform the best or in some cases make the model function. There are many approaches to hyperparameter optimization ,and Yang and Sham [19] present some of them in their paper. 'Grad student descent (GSD)' is one approach to parameter optimization were a persons finds the best parameters based on a trail and test approach. Hyperparameter optimization (HPO) frameworks are automating the process for parameter tuning. GridSearchCV function from Sklearn is an example of HPO framework that test a set of specified hyperparameters, that discover the best performing hyperparameters for ML algorithm [19]. GridSearchCV was implemented on the XGBoost model with feature selection, using the parameters in table 1.

3.4 Implementing MLP

There are multiple approaches to create multi-layered perceptions in Python. A MLP model can have many hidden layers and each layer could have many nodes. For this project the number of layers have been kept to a minimum to keep the complexity relative low and the time limitations. The MLP models were created with some variances. The data used for the model had 13 classes, and therefor the model had 13 nodes in their output layer. All MLP models was created with the selected feature, and subsequently the models had all 17 input nodes, because of the 17 features from the feature selection process.

Sklearn is one library that provides an implementation for an MLP classifier. The library provides a simple library and implantation of the model, with the option for different parameter choices. The MLP model was initially created with a single hidden layer with 20 nodes, but was tested with multiple layers and different amount of nodes for the parameter optimization implementation. The model was created with the rectified linear unit (ReLU) activation function and the Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (LBFGS) solver.

Hyperparameter optimization was implemented for this model, using GridSearchCV function from Sklearn. The hyperparameters that were tested was the activation function, the solver, and the hidden layer size. For activation function the Tanh and ReLU functions were evaluated. The Adam function, LBFGS and stochastic gradient descent (SGD) was proposed for the solver. Three hidden layers architecture was tested, two single layer architecture with 20 and 100 neurons respectively and a two layered architecture with 20 and 10 neurons in the first and second layer respectively.

MLP with TensorFlow & Keras

The second approach to the MLP model was to generate a model with two hidden layers with respectively 128 and 64 nodes using Keras and TensorFlow. TensorFlow and Keras provides built-in functionality for the building and MLP model. The library is used to create the frame for the model using the sequential function. Combined with frame the

Table 2. Model performance

Model	Training Time (min:sec)	Recall	Precision
XGBoost on the full dataset	20:47	0,9878	0,9577
XGBoost with feature selection	01:36	0,8621	0,8565
MLP with Sklearn	03:52	0,8587	0,8537
MLP with TensorFlow Keras	03:49	0,8606	0,8558
MLP from scratch	01:57	0,7070	0,5909

library provides easy implementation of layers for the model. This MLP model was created with the hidden layers using the ReLU activation function, while the output layer uses the softmax activation function. The Adam optimizer, categorical cross-entropy for calculating loss and a learning rate of 0,001 was used. The model was trained over 10 epochs using the train subset and 20% of the training data was selected for validation of the model.

MLP from scratch

The manually created MLP model have a single hidden layer with two nodes. Figure 4 show a graph visualization of the MLP model. The model takes the data with the selected features as input represented by the 'i' nodes, the input is sent to both nodes in the hidden layer and forwarded to the output layer. The model uses both softmax and ReLU activation functions for forward propagation and the ReLU for backward propagation. ReLU is used for the calculations in the hidden nodes, while the softmax function is used for the calculations of the output. The model have a learning rate of 1e-7. The weights and biases variables are random values between 0 and 1, and each variable is fitted to the respective layers.

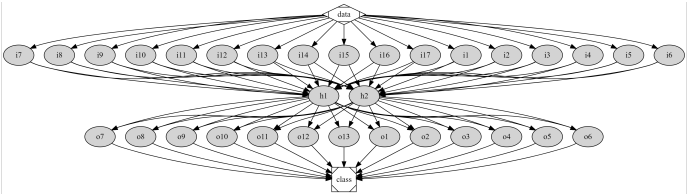

Fig. 4. Graph display of the MLP model from scratch

4 RESULTS

This section will cover the result from each of the models and compare the results and models. The XGBoost model on the full dataset and the model on the dataset with the selected feature will both be analyzed to discover if there difference in performances. The three MLP models will be analyzed and compared, and we will look into the effect of the different approaches as well as the different architecture of the models. Lastly all models will pitted against each other, with a focus on the best performing models of each machine learning algorithm.

Table 2 show the recall, precision and training time for the respective models. Precision and recall is chosen for the study because it provides a better understanding of the models performance in regards to the smallest classes compared to accuracy metric. Additionally to the two metrics we will look into the models classification of the small classes, and how the unbalanced dataset have effected the models and the scores. Training time is the last metric used in the evaluation. The significance of the training time, is highly dependent on the use case. Training time will have less weight in the evaluation of the models, but will be discussed.

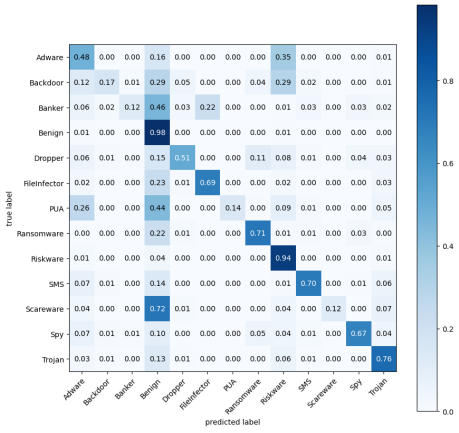

Fig. 5. Matrix for XGBoost model with feature selection predication percentages

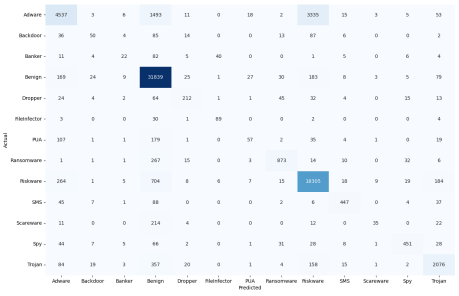

Fig. 6. Matrix for XGBoost model predication with feature selection with number of predictions

The XGBoost model using the data with selected features had an overall good performance. The metrics in table 2 is the recall and precision from the model with the best performing hyperparameters retrieved using the GridSearchCV. The best hyperparameters was root mean squared error as evaluation metric, 0,3 for learning rate, maximum depth of 6, 200 trees and multiclass softmax function. Figure 5 and 6 shows the distribution of predicted classes in contrast with the true classes, with percentages and the number of prediction respectively. The two figures shows that there are bias in the model for the largest classes. A large amount of instances are predicted as Benign or Riskware, which is the two significantly largest classes. The classification of the FileInfector malware category is noteworthy given it is the smallest class it has a large number of true positives.

The XGBoost model created on the whole dataset has the best prediction metrics, as seen in table 2. The model have a 0,98 and 0,95 recall and precision scores respectively. This model is significantly slower in the training process compared to the other XGBoost model. Figure 7 show the distribution of the predicted classes by percentage. From the matrix we can see significant increase in the true positive prediction. There are more prediction in the in the upper section, represented by the dark blue, additionally there are a significant decrease in false positive for the benign class. There generally a reduction in false positive for this model, but there are some false positive, and the Adware and Riskware are most represented in this model also.

The MLP model using Sklearn achieved both recall and precision scores of approximately 0,85. The model is the created based on the best performing hyperparameters from the HPO test. The matrix in figure 8 shows the distribution of predicted classes compared the the actual classes in percentage. The matrix show the largest classes, Benign, Adware and Riskware, have a significant amount of false positives. The scareware and PUA categories have only 10% correctly classified instances out of all actual instances of the respective classes, also the backdoor and file-infector was less than 20% correctly classified instances.

The hyperparameter optimization functions result was that the Tanh activation function was better than the ReLU for this model, and Adam was ranked as a better solver than LBFGS and stochastic gradient descent (SGD). For the three hidden layers architecture that was tested, the GridSearchCV function classified the single hidden layer with 100 neurons as the best architecture. The hyperparameter optimization test took a total of 49 minutes.

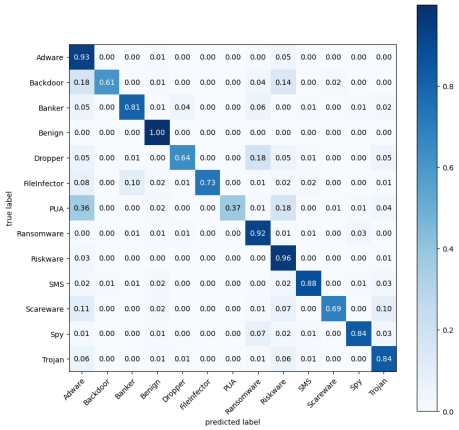

Fig. 7. Matrix for XGBoost on the full dataset with predication percentages

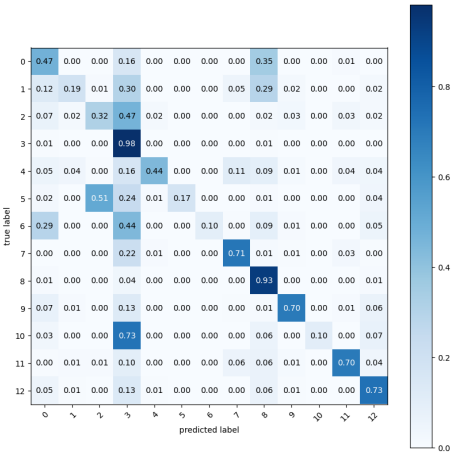

Fig. 8. Matrix for Sklearn MLP with predication percentages

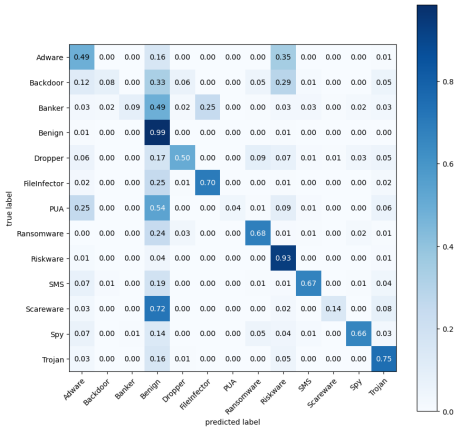

Fig. 9. Matrix for TensorFlow MLP model predication percentages

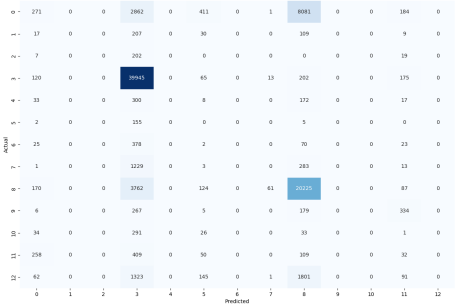

Fig. 10. Matrix for MLP model from scratch's number of predictions

MLP model using TensorFlow and Keras was implemented using GPU computations. This model achieved a precision and recall scores of 0,85 and 0,86 respectively. Those are relative good scores for the classification problem. Figure 9 shows that distribution of predictions was relative even with the class distribution in mind. The backdoor, banker, PUA and scareware classes malware categories had almost zero percent false positives, but also a very low rate of true positives, given that almost no predictions was made of these classes. The Adware class had less that 50% true positives of all actual positives. This model had better performance for classifying some of the smaller classes like FileInfector and Trojan-SMS. The significantly largest class, benign, are significantly most predicted and has many false positives.

The MLP model created from scratch without using i python library had the lowest recall and precision scores of all the models. The model had also the largest difference between the recall and precision score, as you can see in table 2. The Matrix in figure 10 show that the model only predicted the instances as one of six classes out of the twelve. Two of the classes that had been predicted had no correct prediction, meaning all predictions were false positives. The three

largest classes, adware, benign and riskware, had a majority of the predictions as true positives. The last class that was predicted, Trojan-spy had only a minority of the predictions as true positives. The majority of the predicted Trojan-spy malware, were actually Trojan-SMS'.

5 ANALYSIS AND DISCUSSION

Overall the models have relatively good performance metrics when tested. The MLP model created without a library had the significantly worst metrics, as seen in table 2. The XGBoost model that used the full dataset for training had the highest recall and precision by a significant amount, but it also was the model with highest training time. The three other models had a high similarity for all metrics.

The time metric is the metric with the least weight for the analysis and comparison, because the use of such a model does not need frequently training on newly classified applications. Recall and precision has a higher importance for such a model since it informs on the rate the model classifies application correctly. In the analysis we will look into the rate of each model that incorrectly classifies samples, and especially false positive instances of the benign class.

Initially lets examine the two MLP models created using the two python libraries. The precision scores are very close with only a difference of 0,0021 and 0,0019 for precision and recall respectively. It was expected that the TensorFlow MLP model had better performance of the two, as it did, however a larger gap between the two models was excepted. The reason why the TensorFlow model was assumed to outperform the Sklearn model was because of the model had two hidden layers with 128 and 64 neurons. The other model had a single hidden layer with 100 neurons, meaning it had less computations by a significant amount because of MLP is fully connected, and each of the 128 neurons is connected to the 64 neurons in the hidden layers for the TensorFlow model.

There are two main reasons that Sklearn model performance is this close to the TensorFlow model. The first reason is the Sklearn model is implemented with hyperparameter optimization as explained in 3.4. HPO found the optimal hyperparameters for the model, out of the specified parameters. HPO was not performed for the TensorFlow MLP model, which can mean the parameters used in the model are suboptimal. The second reason are the Sklearn MLP model ran for 100 epoch in contrast to only 10 for the TensorFlow model. The higher number of epochs or training rounds gives the model more attempts to optimize the weights and biases used in the model.

Comparatively the MLP model created without the direct use of a Python library for the model, was the significantly weakest performance. Given the neural network had a single layer with two neurons it was anticipated to have the weakest scores. The model was trained over the most epochs, with 500. The number training rounds was expected to help increase the performance of the model, but not result in a recall or precision score in the same range as the other models.

XGBoost algorithms have shown very good performance for different machine learning problems. Nielsen [13] in his thesis, referred to the algorithm as winning "every" ML competition. Based on the performance XGBoost have had for classification problems, it was expected the XGBoost model have high performance. The XGBoost model trained on the features selected dataset have similar precision and recall scores as the two best performing MLP models. The XGBoost model have marginally better recall and precision, with a difference of 0,0034 higher recall and 0,0027 higher precision compared to the two mentioned MLP models.

The XGBoost model trained on the full dataset was the clearly best performing model with 0,98 recall and 0,95 precision. High results were anticipated by the algorithm based on the scores it had displayed in ML competitions. The model had also the significantly highest training time by all the models. The classification problem the model solves, does not mean the model needs to be retrained frequently the training time is not an issue. In addition to the

two mentioned metrics the most noteworthy result is the falsely classified instances. The model has a significantly low number of false positives for the benign class, as seen in figure 7. The model not classifying malware as benign is a especially important feature for a model solving this classification problem. False positives for the benign class will mean that a malicious application could be downloaded or stay present at a device without the user knowing it is malicious.

The full dataset XGBoost model was the best overall model. The low false positive rate for the benign class was the most key feature of the model compared to the other models. The XGBoost model outperforming the other models on both precision and recall was another factor. The only malware category the model had a lower true positive percentage than 60 was, was for potentially unwanted application (PUA). There are a large variety between PUAs, and therefor it can be expected to be hard to classify. From figure 7 we can see that this XGBoost model is effected by the unbalances in the dataset, for especially the Adware category have significant of false positives. Actual PUA files are the most represented false positives for the predicted Adware category. The malware categories classes that are most represented in the dataset do have more false positives compared to the smallest classes. The assumed reason for this XGBoost model performance was better comparatively to the XGBoost trained on the feature selected data, was the built in feature selection in the model. The built in feature selection have made the model find the optimal feature for each tree, while with the feature reduction some of the optimal features could have been removed.

6 CONCLUSION

In conclusion, this study delved into the use of machine learning for classifying Android malware categories, with an implementation of MLP and XGBoost. The aim was to analyze and compare the two approaches, and observe any differences in performance. Through our study we delved into the five created ML models with their respective performance metrics.

The key finding showed that the XGBoost model trained on the full dataset was the clearly best performing model, with the highest recall score with over 10 percentage points to the second highest scoring model. The MLP models with more neurons performed at the same rate as the second best XGBoost model, while the MLP model with a single layer with two neurons had the poorest scores. The difference between the MLP models showed the importance of a high number of neurons and hidden layers, combined with a high number of training rounds. The XGBoost model revealed the quality of combining multiple weaker learners, and the quality builtin feature selection in the algorithm. The most important metric from the XGBoost model was the low number of false positives for the benign class. This means the model classifies a low number of actual malware as a benign application. A malware classifier's most important feature for most users are to prevent the download or detect the presence of malicious applications. With a low false positive rate for the benign class the full dataset XGBoost model does this.

REFERENCES

[1] AV-TEST. 2023. *Statistics.* "https://portal.av-atlas.org/malware/statistics"

[2] Ikram Ben Abdel Ouahab, Lotfi Elaachak, and Mohammed Bouhorma. 2022. Image-based malware classification using multi-layer perceptron. In *Networking, Intelligent Systems and Security: Proceedings of NISS 2021*. Springer, 453–464.

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[4] David Curry. 2023. Android Statistics (2023). https://www.businessofapps.com/data/android-statistics/

[5] Statista Research Department and Oct 4. 2023. Mobile OS market share worldwide 2009-2023. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[6] Junxi Feng, Xiaohai He, Qizhi Teng, Chao Ren, Honggang Chen, and Yang Li. 2019. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E* 100, 3 (2019), 033308.

[7] Matt W Gardner and SR Dorling. 1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment* 32, 14-15 (1998), 2627–2636.

[8] David Sean Keyes, Beiqi Li, Gurdip Kaur, Arash Habibi Lashkari, Francois Gagnon, and Frédéric Massicotte. 2021. EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*. IEEE, 1–12.

[9] Igor Kononenko and Matjaz Kukar. 2007. *Machine learning and data mining*. Horwood Publishing.

[10] Ioannis Kouretas and Vassilis Paliouras. 2019. Simplified hardware implementation of the softmax activation function. In *2019 8th international conference on modern circuits and systems technologies (MOCAST)*. IEEE, 1–4.

[11] Huan Liu and Rudy Setiono. 1998. Incremental feature selection. *Applied Intelligence* 9 (1998), 217–230.

[12] Johan Luhr and Hannes Hallqvist. 2022. Fast Classification of Obfuscated Malware with an Artificial Neural Network.

[13] Didrik Nielsen. 2016. *Tree boosting with xgboost-why does xgboost win" every" machine learning competition?* Master's thesis. NTNU.

[14] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. 2009. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems* 8, 7 (2009), 579–588.

[15] Yanchen Qiao, Weizhe Zhang, Xiaojiang Du, and Mohsen Guizani. 2021. Malware classification based on multilayer perception and Word2Vec for IoT security. *ACM Transactions on Internet Technology (TOIT)* 22, 1 (2021), 1–22.

[16] Abir Rahali, Arash Habibi Lashkari, Gurdip Kaur, Laya Taheri, Francois Gagnon, and Frédéric Massicotte. 2020. Didroid: Android malware classification and characterization using deep image learning. In *2020 The 10th international conference on communication and network security*. 70–82.

[17] Mohammed Rashed and Guillermo Suarez-Tangil. 2021. An analysis of android malware classification services. *Sensors* 21, 16 (2021), 5671.

[18] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. 2017. Activation functions in neural networks. *Towards Data Sci* 6, 12 (2017), 310–316.

[19] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316.

Received 15 December 2023; revised ; revised