Analysis of Malware Prediction in Windows Devices

Mohammad Anas, Ying Feng, Vicki Nomwesigwa, Deekshita Saikia
(Team 9)

Final Report

April 14th 2022

Abstract

Malware infection rates have been on the rise in recent years. Given the data deluge the industry currently faces, it is essential to guard our devices well so that the impact from malware attacks can be minimized. Through this analysis, we explore different tree-based models that can predict a Windows machine's probability of getting infected by malware, based on different hardware and software configurations of that machine. A LightGBM classification model performed the best in our hold-out dataset in predicting if a machine would be infected. We also delve into which features of a device are most sensitive to a malware attack, and it was found that the diagonal display size is particularly sensitive to changes in the underlying sample.

Introduction

Malware, short for malicious software, are intrusive software designed to damage and destroy computers and computer systems. These software come in various forms, such as trojan horses, computer viruses, worms, ransomware, and spyware. According to the 2021 cyber-security analysis conducted by *PurpleSec*, a veteran-led cyber security company based out of Washington, DC, it was reported that 34% of businesses that get hit with malware took a week or more to regain access to their data (PurpleSec, 2021). Malware infections have risen from 12.4 million since 2009 to 816.67 million in 2018, and, therefore, device protection from malware is paramount. This analysis seeks to predict the probability of a Windows machine getting infected by malware. We also seek to understand the features of such machines that make it more prone to malware attacks.

This paper is organized as follows: in the background section, we discuss related work; in the data section, we present our data sources and exploratory data analysis; in the methods section, we present the pre-processing steps and the various models considered; in the results section, we discuss the performances of the different models and experimental analysis results; finally, in conclusion, we analyze our results and present possible future research.

Background

Several researchers have focused on different malware detection techniques to combat the rising numbers of malware infections. These different techniques fall into two categories: static malware detection and dynamic malware detection (e.g., (S. Cesare, 2013; H.S. Galal, 2016)). Static malware detection analyzes malware samples without execution of the malware software itself, thereby allowing analysts to observe the behavior of the malware and determine what it is through the extraction of program features (Ekta Gandotra, 2014). One limitation of static malware detection methods is that they are insufficient, and using confusion will evade them (e.g., (Daniele Uccia, 2018; P. Burnap, 2018)). On the other hand, dynamic malware detection executes suspected malicious code in a sandbox that is a safe environment. The execution involves extracting features such as network behavior, registry change, system calls, and memory usage (Y. Qiao, 2014). Dynamic methods can efficiently analyze packed and obfuscated malware (W. Han, 2019; I. Zelinka, 2019).

In contrast, many recent documented works use machine learning algorithms to detect malware in mobile devices and computers. Researchers like (Bryan Dixon, 2011), (Hahnsang Kim, 2018), and (Lei Liu, 2009) have used power analysis to detect malicious software in smartphones, and their findings indicate anomalous consumption of resources. Others like (Sanjeev Das, 2016) used the n-Gram data mining technique to extract system calls and create features based on the high-level semantics of malicious behavior. Sanjeev et al. showed that "GuardOL," hardware that detects malware at runtime, was fast and efficient and had a performance overhead on processor cores. However, the majority of these analyses rely on using malicious binary files and their meta-data and code to identify malware. In this analysis, we will focus on using tree-based classification methods to predict if a device is likely to be infected by malware, leveraging solely hardware and software configurations of the device. We also strive to explore which specifications of a device are strongly associated and sensitive to malware.

Data

The dataset used in this analysis was collected from Kaggle (Kaggle, 2022), provided by Microsoft to encourage open-source progress on effective techniques for predicting malware occurrences. The goal of this competition was to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. The telemetry data containing these properties and the machine infections was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender.

The original dataset has 16 million values and 83 features, most of which were categorical attributes. Each row in this dataset corresponds to a machine, uniquely identified by a `MachineIdentifier`. `HasDetections` is the ground truth and indicates if malware was detected on the machine. A significant limitation of this analysis was the size of the datasets, which warranted significant compute and time resources. Owing to these constraints, we extract a stratified sample of approximately 1.5 million (~17%) observations from the original dataset. The stratification was performed on `HasDetections`, `Platform`, `Processor`, `IsProtected` and `Census_IsTouchEnabled`. The codebook can be referred to here.

Before proceeding to pre-processing, the dataset is split into 70 – 30 train-test samples. This training sample is further split into 80 –20 train and validation samples, for the purpose of hyper-parameter tuning. We fit the models on the training dataset from the train-validation split, not the train-test split.

In this analysis, we are interested in predicting the probability of a Windows machine getting infected by malware based on the different properties of the machine. Figure 1 shows that the fraction of machines that had malware detections and those that did not are comparatively similar.

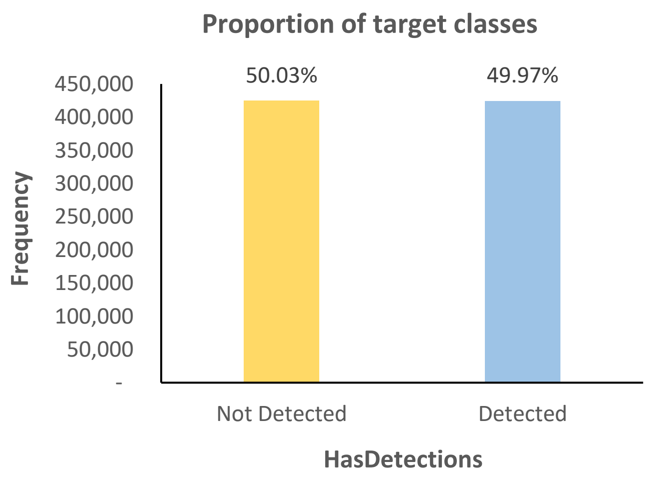

Figure 1: Proportions of observations in each target class

In the data cleaning process, we dropped all features that had at least 50% of their values as nulls. We imputed the null values of remaining categorical and numerical columns by replacing them with the string "*Missing*" and the median respectively. This is done as we wanted to analyze the null values for the categorical attributes as a separate category.

The exploratory analysis of numerical attributes revealed significant outliers, as shown in the left plot of Figure 2. For these attributes, we replaced the outliers using the three-sigma rule of thumb. The distributions of the raw numerical attributes were skewed, hence we also considered log transformations of these attributes. The plot on the right of Figure 2 shows the resulting plots after outlier treatment and log transformations.

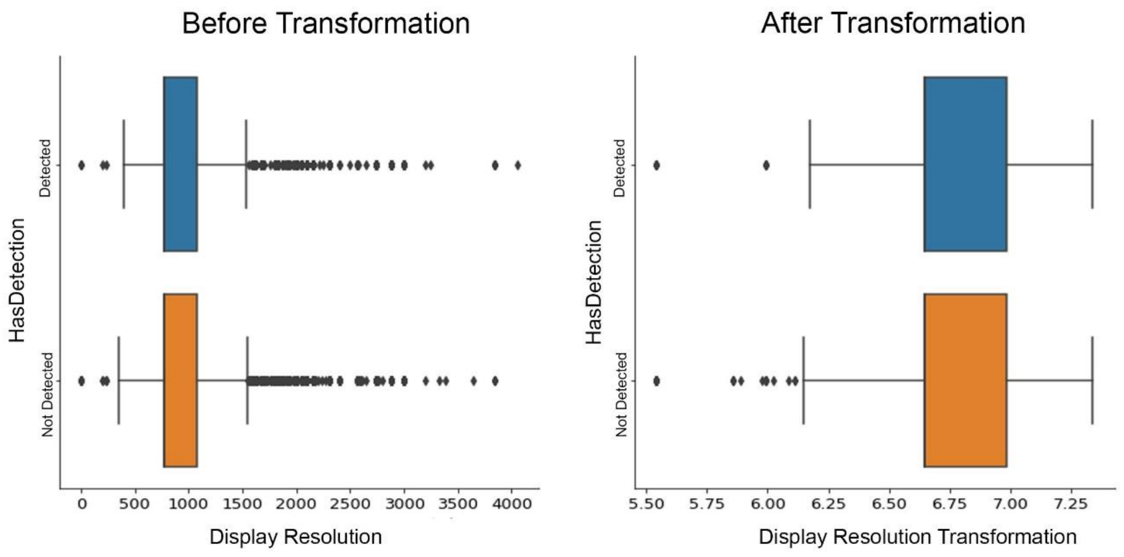

Figure 2: On the left, the boxplot corresponds to the distribution of Display Resolution, and on the right, the box plot shows the distribution post removal of outliers and log transformation.

It is worthwhile to note that the sampling methodology used to create this dataset was designed to meet certain business constraints, both with regards to user privacy as well as the time period during which the machine was running. Malware detection is inherently a time-series problem, but it is made complicated by the introduction of new machines, machines that come online and offline, machines that receive patches, machines that receive new operating systems, etc. Additionally, this dataset is not particularly representative of Microsoft customers' machines in the real-world; it has been sampled to include a much larger proportion of malware machines.

Methods

Correlations are checked amongst pairs of numerical attributes, and a threshold of 70% is set to decide which correlated features to drop. The categorical variables were transformed through one-hot-encoding, to prepare the data for feature selection and classification. We created a customized frequency encoding Python class to compute one-hot-encoded values for categorical features. The class retains the most commonly occurring features based on a given limit. For example, each categorical feature encoded required at least 2,000 values in this analysis.

A random forest classifier is trained on the dataset to rank each feature based on their importance, which is calculated from the information gain achieved by splitting the data on that feature. A cross-validation process was performed by measuring the AUC scores on a sample of the training data to iteratively select the number of features (Qi, Qiu, & Diao, 2019). The cross-validation plot for feature selection is as shown in the flowchart in Figure 3.

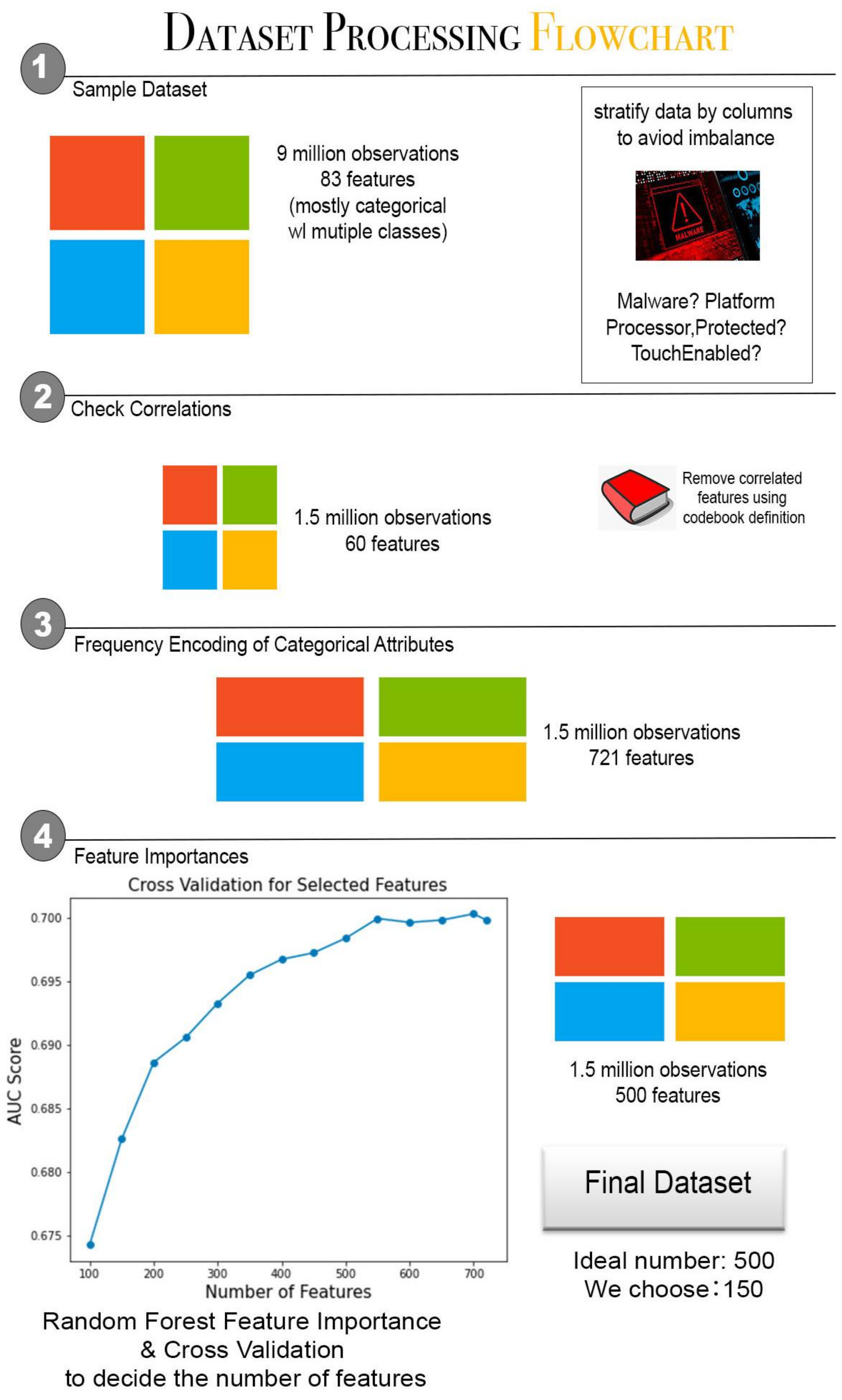

Figure 3: Data pre-processing pipeline

The feature importances obtained from the random forest classifier is as shown in Figure 4 below.

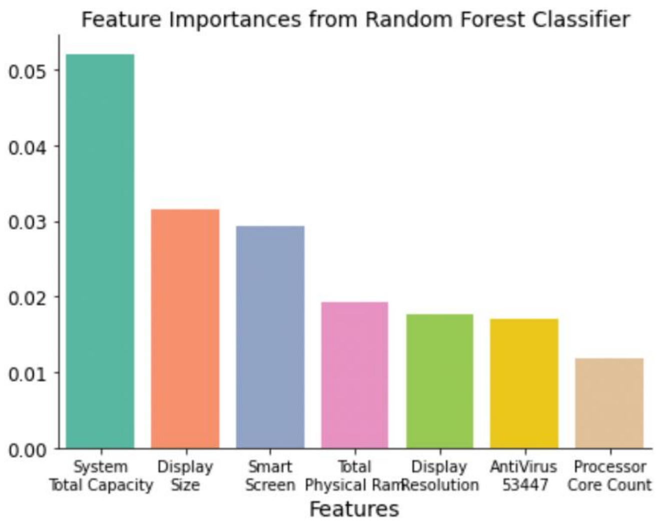

Figure 4: Features importances from Random Forest classifier

To build our predictive model, we rely on tree-based models because of the large number of categorical features in our data and the skewed distribution of our numerical columns. We fit four different classifiers on our train data set: a decision tree classifier which was used as our baseline model, Random Forest classifier, XGBoost classifier, and a LightGBM classifier. We used Bayesian optimization to tune the hyperparameters of these classifiers. As opposed to grid search and random search, this optimization keeps track of past evaluations and leverages them to search for hyper-parameters that optimize the objective function, which in our case was equivalent to maximizing the AUC score on our validation dataset. This allowed us to tune hyper-parameters more efficiently given the constrained power of our machines.

Our choice of metric to compare across models is the ROC AUC, which is a standard in classification problems. We then use our best performing predictive model to dive deeper into exploring the association of some important features in our dataset with our response variable.

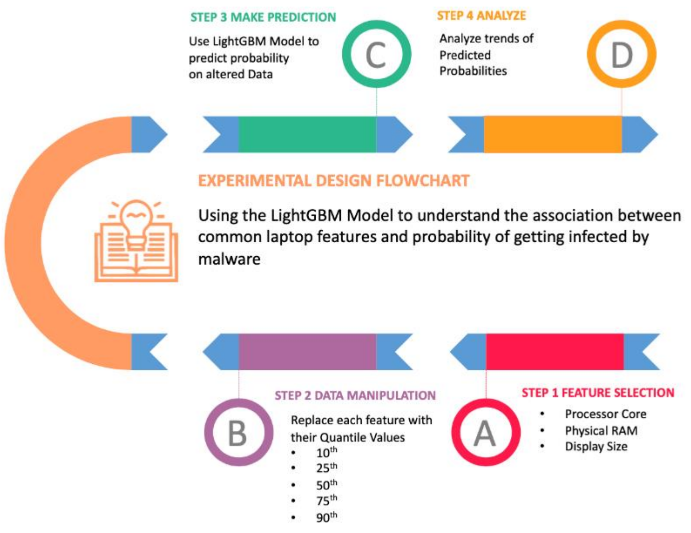
STEP 3 MAKE PREDICTION
Use LightGBM Model to predict probability on altered Data
C
STEP 4 ANALYZE
Analyze trends of Predicted Probabilities
D
EXPERIMENTAL DESIGN FLOWCHART
Using the LightGBM Model to understand the association between common laptop features and probability of getting infected by malware
STEP 2 DATA MANIPULATION
Replace each feature with their Quantile Values
10th
25th
50th
75th
90th
B
STEP 1 FEATURE SELECTION
Processor Core
Physical RAM
Display Size
A

Figure 5: Flowchart for experimental design

For our experiment, we use the idea of "No Causation Without Manipulation" (Holland, 1986; Rubin, 1978). We manipulate our dataset by replacing all the rows for the feature

`ProcessorCoreCount` with different quantile values corresponding to that feature. Using our trained model, we make predictions on the altered training datasets and analyze the varying trends of average predicted probabilities. We perform the same analysis for several other features in our training sample.

Results

Figure 6 shows a comparison of the ROC and PR curves for the random chance model and the four trained tree-based classifiers. Compared to our baseline model, the decision tree classifier, which has an ROC AUC of 56.85%, we observe that the Random Forest classifier, the XGBoost classifier, and the LightGBM classifiers have 23.29%, 23.59%, 23.71% improvement in the AUC scores respectively. Since the LightGBM classifier performs the best in our hold-out test dataset, we use this as our classifier of choice.

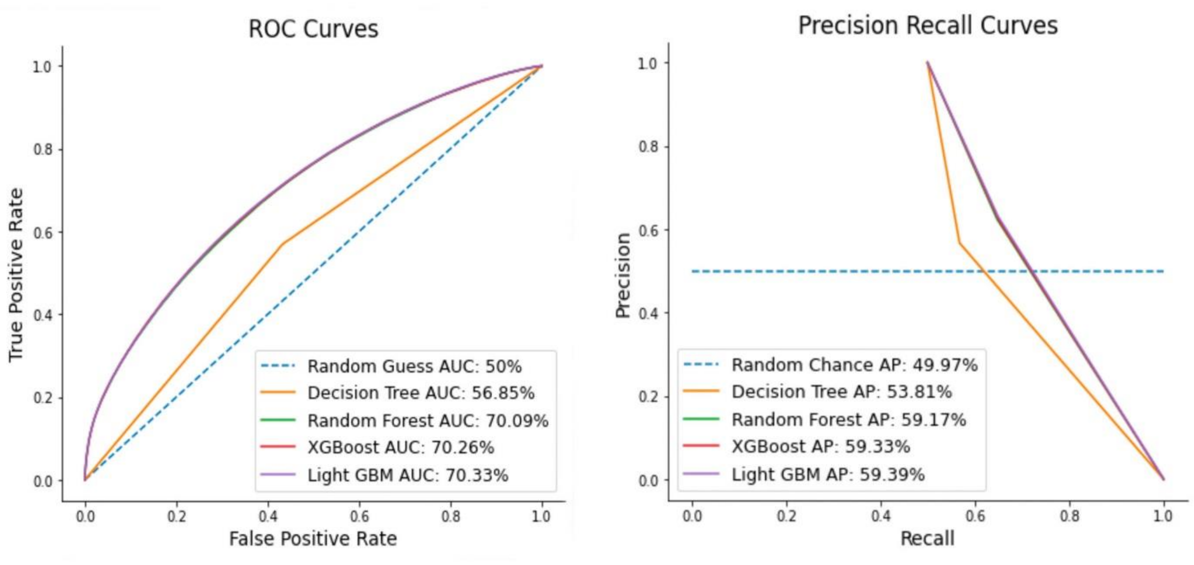

Figure 6: ROC and PR curves for the Decision Tree, Random Forest, XGBoost and LighGBM classifiers.

Figure 1 shows that there is no imbalance in the target classes in our dataset; therefore, accuracy scores can be used to evaluate our models. We observe that our model can correctly classify whether a Windows machine will be infected by malware or not 64.51% of the time. We also note that in our context, the cost of false negatives is higher as compared to false positives. Therefore, we examined our recall score, which is 63.02%. The confusion matrix for this classifier is shown in Figure 7.

Using the Light GBM model, we then make predictions on our simulated datasets. We found out that our model identifies machines with higher RAM to be more prone to malware. The average predicted probability of being infected by malware was also positively associated with an increase in number of processor cores. This association can be seen in Figure 8 below.

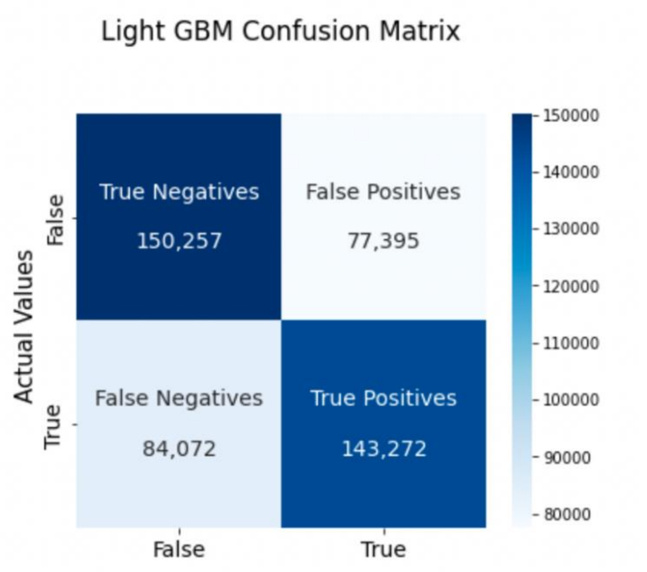

Figure 7: Confusion Matrix of Light GBM model on test dataset

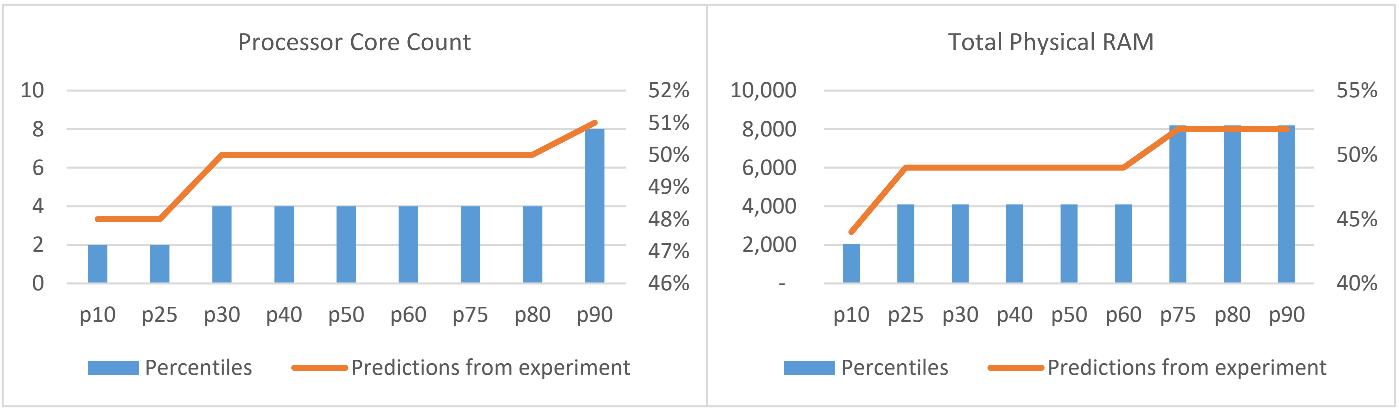

Figure 8: Sensitivity analysis for ProcessorCoreCount and TotalPhysicalRAM.

One potential reason for this association could be attributed to the increase in fileless malwares, which have been notoriously on the rise since 2017, due to its ability to attack devices through hardware features. Certain models of x86 processors contain secondary embedded RISC-like CPU cores that can be bypassed by fileless malwares (Microsoft Docs, 2022). Observing the for the display size, we see that our model is quite sensitive, however an increasing trend was observed for this as well.

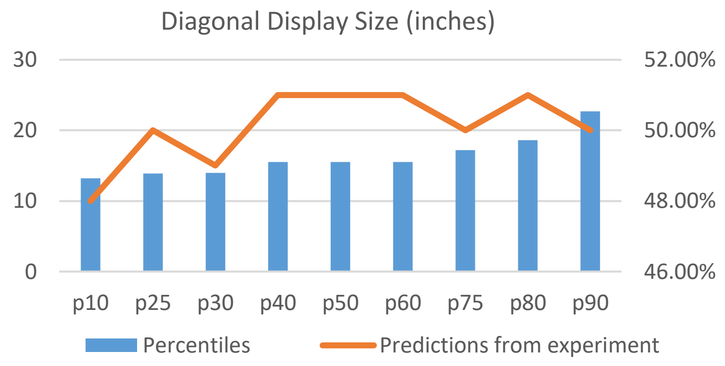

Figure 9: Sensitivity analysis for Diagonal Display Size

Conclusion

With the increased need to protect devices against malware attacks, this analysis explores various tree-based classification models to predict if a device is likely to get infected by malware, using various hardware and software specifications from a device. This analysis relies on data solely from device specifications, as opposed to existing studies which leverage meta-data and code from malicious binary files. To avoid high dimensionality while modelling, we examined feature importances from a random forest classifier, and choose the best 150 features for model training. We compare four different tree-based classifiers, out of which the LightGBM classifier records the highest ROC AUC score in our test dataset. We also delved deeper to examine if predictions of malware attacks are especially sensitive to certain device attributes. Based on our sensitivity experiments, we observe that our model is highly sensitive to diagonal display sizes of devices, and moderately sensitive to physical RAM and number of processor cores in the device.

Roles

- Mohammad Anas, *Programmer*: Responsible for programming the models, and experiment design. Also assisted Deekshita and Ying in exploratory analysis of the dataset and data pre-processing.
- Ying Feng, *Programmer:* Responsible for programming the exploratory analysis piece. Also responsible for creating flowcharts and maintaining visualizations for the analysis.
- Vicki Nomwesigwa, *Writer*: Responsible for managing the Github repository and assisting with data engineering. Also responsible for coordinating team deliverables and writing the first draft of the report.
- Deekshita Saikia, *Writer:* Responsible for programming the data pre-processing pipeline. Also responsible for creating the presentation slides and editing the final draft of the report.

References

1. Bryan Dixon, Y. J. (2011). Location based power analysis to detect malicious code in smartphones. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM*, 27-32.
2. Daniele Uccia, L. A. (2018). Survey of machine learning techniques for malware analysis. *Computer Security*.
3. Ekta Gandotra, D. B. (2014). Malware analysis and classification: a survey. *Journal of Information Security*, 56-64.
4. H.S. Galal, Y. M. (2016). Behavior-based features model for malware detection. *J. Comput. Virol. Hacking Tech.*, 59-67.
5. Hahnsang Kim, J. S. (2018). Detecting energygreedy anomalies and mobile malware variants. *Proceedings of the 6th international conference on Mobile systems, applications, and services, MobiSys*, 239–252.
6. Holland, P. W. (1986). Statistics and Causal Inference. *J. Am. Stat. Assoc*, 945-60.
7. I. Zelinka, E. A. (2019). An ensemble-based malware detection model using minimum feature set. *Mendel, 25 (2)*, 1-10.
8. Kaggle. (2022, April 06). *Microsoft Malware Prediction*. Retrieved from Kaggle: https://www.kaggle.com/c/microsoft-malware-prediction
9. Lei Liu, G. Y. (2009). Virusmeter: Preventing your cellphone from spies. *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection, RAID*, 244–264.
10. Microsoft Docs. (2022, April 6). *Fileless threats*. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/microsoft-365/security/intelligence/fileless-threats?view=o365-worldwide
11. P. Burnap, R. F. (2018). Malware classification using self organising feature maps and machine activity data. *Computer Security*, 399-410.
12. PurpleSec. (2021, October 28). *10 Cyber Security Trends You Can't Ignore In 2021*. Retrieved from https://purplesec.us/cyber-security-trends-2021/
13. Rubin, D. B. (1978). Bayesian Inference for Causality: The Importance of Randomization. *The Annals of Statistics 6*, 34-58.
14. S. Cesare, Y. X. (2013). Control Flow-Based Malware VariantDetection. *IEEE Trans. Depend. Secure Comput.*, 307-317.
15. Sanjeev Das, Y. L. (2016). Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Malware. *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*.
16. W. Han, J. X. (2019). MalDAE: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Comput. Secur., 83*, 208-233.
17. Y. Qiao, Y. Y. (2014). CBM: free, automatic malware analysis framework using API call sequences. *Knowledge Engineering and Management, Springer, Berlin*, 225-236.
18. Qi, C., Qiu, L., & Diao, J. (2019). *On Estimating Model in Feature Selection With Cross-Validation*. IEEE. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8610148&tag=1