Predicting Device Vulnerability via Machine Learning

Brandon Gray, Md Sanzid Bin Hossain, Nabila Shahnaz Khan
University of Central Florida

Abstract

Many innovations have positively contributed to the modern era of computing. However, these legitimate innovations can be exploited by malicious users, or adversaries. There are currently many methods for adversaries to attack computing systems, but a common method is to leverage malicious software, which is labeled as malware. To combat attacks utilizing malware, Microsoft created a Kaggle competition to utilize machine learning and data science concepts to determine whether a device is susceptible to becoming affected by various families of malware. Our project team took interest in this competition and created a Super-Learner model to participate in the competition. The base models within our overall Super-Learner model achieved approximately 68 percent, which is on par with the higher positions of the competition's leaderboard.

1 Introduction

In the era of modern computing, cybersecurity has become an important area of study and research due to the increasing abundance of threats and the alarming rate of attacks being launched by malicious users, who are commonly referred to as adversaries. Many adversaries possess a wide repertoire of attacks aimed at achieving personal or financial gain for themselves. One method of attack involves an adversary executing malicious code on computing systems. These malicious programs are collectively labeled as malware. Microsoft, a preeminent entity within enterprise and consumer computing, launched a Kaggle competition to address this issue in December 2018, called "Microsoft Malware Prediction." The objective of the competition was to utilize data science and machine learning techniques to determine whether a particular device is susceptible to becoming affected by various families of malware. As part of this competition, Microsoft provided anonymized training and testing data sets which contained metadata from real devices. This metadata was created by combining heartbeat and threat reports collected by Windows Defender. Understanding the magnitude of the competition Microsoft created, our project team took particular interest and joined the competition with the intention of creating accurate machine learning models to address these important issues within cybersecurity.

2 Background

Due to increased public use of the Internet, malware is becoming more widespread day by day. According to AV-Test, there were about 99.71 million malware programs found in 2012. In 2020, there were more than thousands of millions malware found [1]. There are over 350,000 new malware created every day. According to recent study, there was a total of 17.85 million new malware. In summary, the malware industry is expanding rapidly, leaving the whole IT industry at great risk.

2.1 Problem Statement

Within the area of cybersecurity, malware is an important issue due to its abundance and severity. In response, Microsoft, a preeminent entity within enterprise and consumer computing, challenged the machine learning and data science communities to address this issue by launching a Kaggle competition in December 2018, called "Microsoft Malware Prediction." The goal of this competition is to create a machine learning classifier to predict whether a device is vulnerable to malware or not based on metadata collected from other devices [2]. The device metadata provided by Microsoft was generated by combining real (anonymized) heartbeat and threat reports collected by Windows Defender. As machines go online and offline during different time periods and new machines become active, an interesting aspect of this competition is to consider the time-series nature of malware detection [2]. Thus, the provided dataset was roughly split by time.

2.2 Super-Learner Base Learners

2.2.1 Gradient Boosting Classifier

The concept of gradient boosting was introduced in a paper titled "Greedy Function Approximation: A Gradient Boosting Machine" by Jerome H. Friedman in February 1999. The gradient boosting concept aims to solve the function estimation (also known as the "predictive learning") problem by taking multiple weak learners and combining them into a stronger learner. The overarching goal of gradient boosting is to find an approximation function of a function that maps training samples to predictions while minimizing a specified loss function over a joint probability distribution of all training samples and predictions [6]. This paradigm has been modified by other authors to create different forms of gradient boosting classifiers, as seen in [8], [3], [4], and [5]. Therefore, gradient boosting is an important theoretical concept within the area of machine learning.

2.2.2 Light Gradient Boosting Machine Classifier

The Light Gradient Boosting Machine (LightGBM or LGBM) classifier appeared in the Proceedings of the Thirty-first Conference on Neural Network Information Processing Systems (NeurIPS) in 2017. The paper was authored by Guolin Ke, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu from Microsoft Research, as well as Qi Meng from Peking University and Thomas Finley from Microsoft Redmond. LightGBM aims to reduce the number of data instances and the number of features in a dataset to improve the performance in a variety of applications, such as multi-class classification, click prediction, and learning to rank [8]. Towards this goal, LightGBM introduces and incorporates two novel techniques: Gradient-based One-side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS is a sampling technique which can reduce the number of data instances as well as maintaining the accuracy of the base decision trees, while EFB is a technique designed to reduce the number of features within the dataset [8].

2.2.3 XGBoost Classifier

The XGBoost classifier was introduced in the paper titled "XGBoost: A Scalable Tree Boosting System," which appeared in the Proceedings of the Twenty-second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. The paper was authored by Tiangqi Chen and Carlos Guestrin from the University of Washington. The XGBoost classifier is a scalable gradient boosting classifier which utilizes decision trees as base classifiers. To provide end-to-end scalability, XGBoost incorporates a novel sparsity-aware algorithm for parallel tree learning as well as a novel weighted quantile sketch for approximate tree learning [3]. Additionally, XGBoost introduced an effective cache-aware block structure to perform out-of-core tree learning [3].

2.2.4 CatBoost Classifier

The paper introducing the CatBoost classifier appeared in the ML Systems Workshop as part of the Thirty-first Conference on Neural Information Processing Systems (NeurIPS) in 2017. The paper was authored by Yandex researchers Anna Veronika Dorogush, Vasily Ershov, and Andrey Guilin. The CatBoost classifier is like other gradient boosting classifiers, as CatBoost utilizes decision trees as base predictors, but CatBoost can perform gradient boosting with categorical features during training as opposed to during preprocessing [4]. Furthermore, CatBoost utilizes a new schema to calculate leaf values when a decision tree structure is selected to help reduce overfitting [4]. CatBoost can handle categorical features by utilizing one-hot encoding, which converts categorical features into numerical features, as well as calculating different statistics using the label values of the examples. CatBoost is comprised of two algorithms: a learning algorithm and a scoring algorithm. Additionally, the CatBoost classifier has a hybrid implementation, where its learning algorithm has a GPU implementation, and its scoring algorithm has a CPU implementation. To begin the prediction process, the CatBoost classifier utilizes oblivious trees as base classifiers [4]. Then, CatBoost performs feature discretization into a fixed number of bins to utilize a histogram-based approach to find the best splits to classify the data [4].

2.2.5 AdaBoost Classifier

The AdaBoost classifier was introduced by Yoav and Robert Schapire in 1995. The overarching principle behind the AdaBoost classifier involves combining many weak and inaccurate classifiers into a highly accurate classifier. AdaBoost implements this functionality by maintaining a distribution of weights over the training set [5]. During execution of the AdaBoost algorithm, weak classifiers find weak hypotheses with low error. In this context, a weak classifier can produce predictions that are considered better than random predictions. After each weak classifier produces a hypothesis, an importance measure is

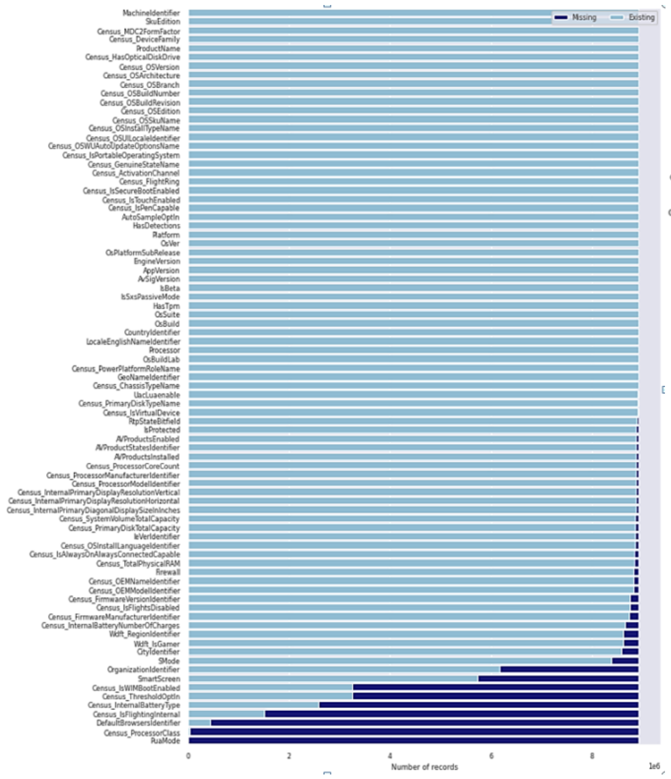

Figure 1: Distribution of missing values against each feature

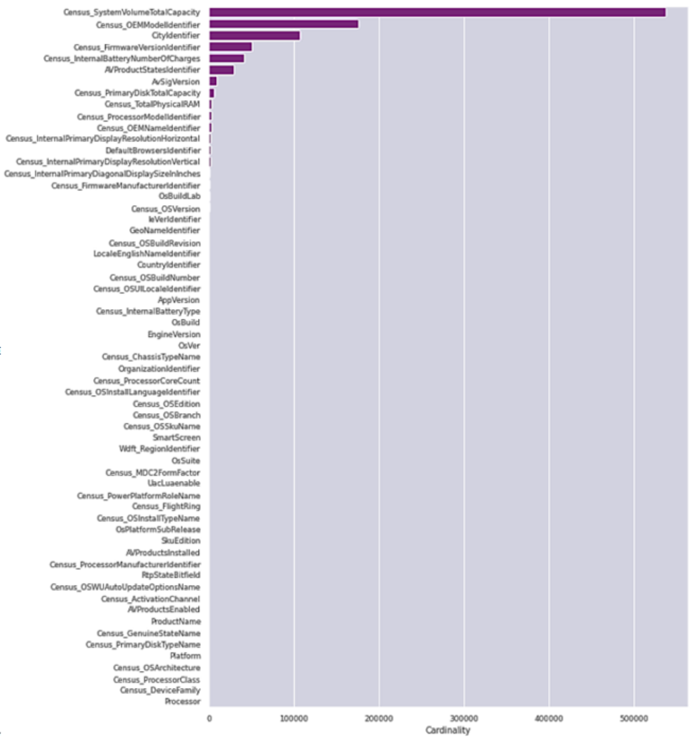

Figure 2: Distribution of cardinality against each feature

assigned to each weak hypothesis generated by each weak classifier. Subsequently, the weight distribution is updated to increase weights of misclassified hypotheses and decrease the weights of correctly classified hypotheses. Finally, a final hypothesis is generated by having each weak classifier take a weighted majority vote to produce the final prediction. AdaBoost is a flexible classifier since no prior knowledge of each weak classifier is required and no parameters, except the number of rounds, have to be specified [5].

3 Dataset

The dataset used in this project was provided by Microsoft and hosted by Kaggle [2]. It was derived from heartbeat and threat reports created by Windows Defender. The training dataset is comprised of almost 9 million data objects. Total size of the uncompressed training data is approximately 4.08 GB. Each row in this dataset corresponds to a machine, uniquely identified by a 'MachineIdentifier'. Here total number of features is 83 and data can be divided into two class labels 'Has detected malware' and 'Has not detected malware'. The dataset consists of features like ProductName, EngineVersion, AppVersion, CountryID, CityID, OrganizationID, platform etc. The column 'HasDetection' shows the class label of the objects. In total there are 53 numerical features and 30 categorical features.

4 Dataset Analysis

Among all the features, 44 features contained missing values and 61 features had cardinality > 2. All these features were separated into three categories: Binary, Categorical, and Nominal where nominal features containted all the numerical values, binary features had a cardinality of exactly 2 and categorical features have cardinality > 2. After the segregation, there were 8 numerical features, 21 binary features and 54 categorical features. The missing value and cardinality distribution of each feature has been shown in Figure 1 and 2 respectively. The dataset is quite well balanced as shown in Figure 3, containing 4458892 rows of has detected malware and 4462591 rows of has not detected malware.

In order to to understand the relation among all the features, feature correlation values have been generated. The feature correlation heatmap has been shown in Figure 4. As we can see from the figure, none of the features have any

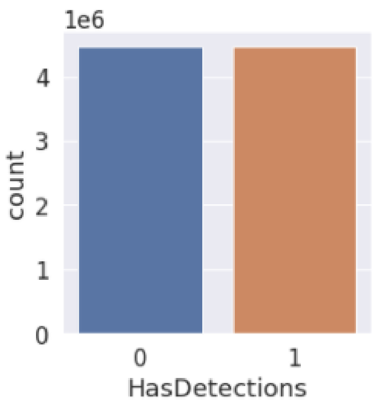

Figure 3: Dataset Balance

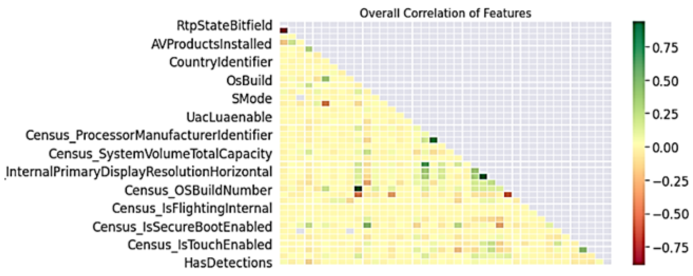

Figure 4: Heatmap showing correlation among all the features

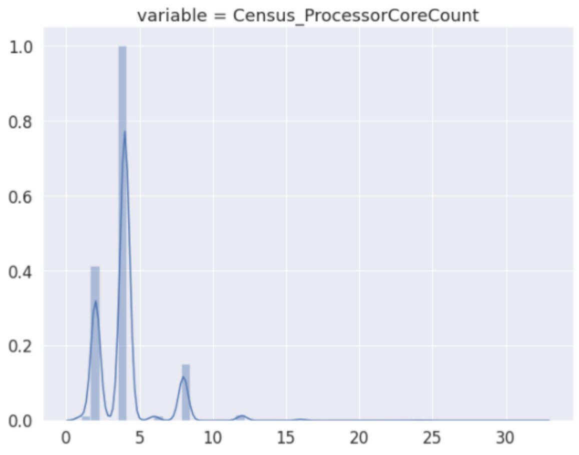

Figure 5: Skewness of feature 'Census Processor Core Count'

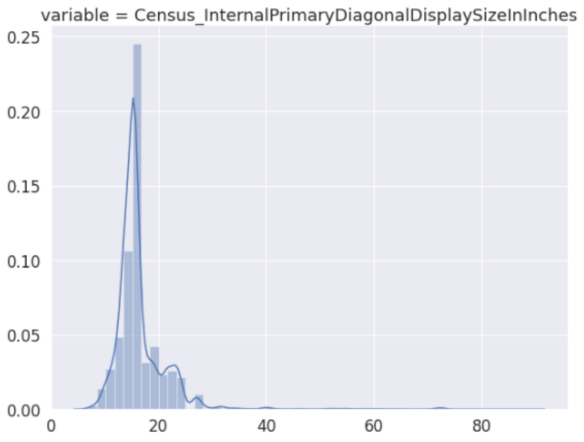

Figure 6: Skewness of feature 'Census Internal Primary Diagonal Display Size In Inches'

significant correlation with the target feature 'HasDetections'. Later, to check if the eight nominal features are normalized or not, skewness of those features have been checked. Only features 'Census_InternalPrimaryDiagonalDisplaySizeInInches' and 'Census_ProcessorCoreCount' are slightly right skewed and have a little high skew value of 5.7 and 4.8 as shown in Figure 5 and 6 respectively.

5 Evaluation Metrics

For evaluating the performance of the model's, we considered four metrics .i.e., overall accuracy, precision, recall and f1 score. If t_p, f_p, f_n is true positive, false positive, and false negative respectively, then

$$\text{precision}=\frac{t_p}{t_p+f_p}$$
$$\text{recall}=\frac{t_p}{t_p+f_n}$$
$$\text{f1-score}=\frac{2\times precision\times recall}{precision+recall}$$

6 Methodology

In this section, we will the discuss proposed workflow for our algorithm. In Figure 7, we showed the basic components of the algorithm. At first, we passed the entire dataset on the pre-processing block. Then, we performed feature selection based on multiple models. Then, we train multiple model on those pre-processed and selected features. Based on the model's

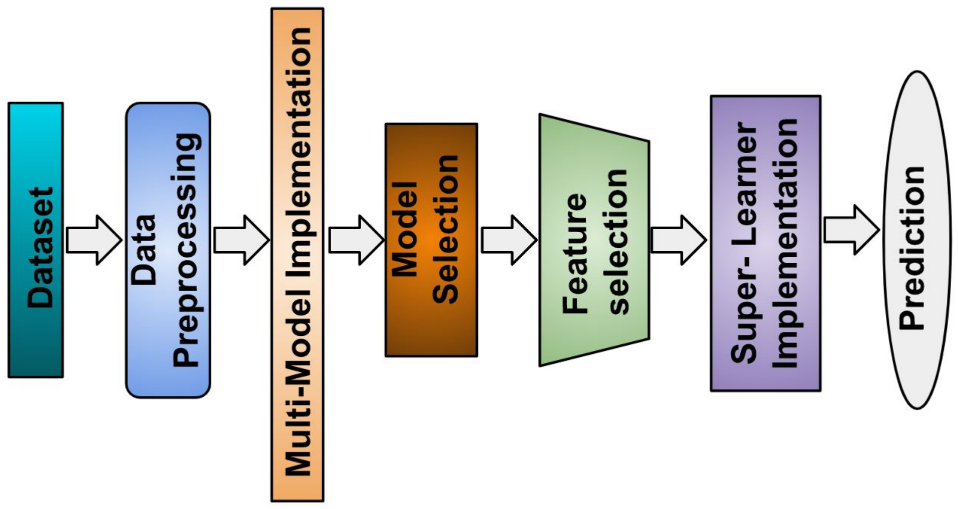

Figure 7: Proposed Workflow

performance both with respect to accuracy and runtime, we select five high performance model and passed it to the Super-Learner for further performance improvement. In the following subsection, we will discuss all the block of the proposed workflow to explain our work clearly.

6.1 Data Pre-processing

We pre-processed approximately 9 million entries consisting of 83 features, where most features consist of nominal data while other features consist of ratio data. We dropped rows associated with numerical features which contained missing values and we converted categorical features to numerical features by using a label encoder. We found 44 columns containing missing values, dropped 10 features with high cardinality, and dropped 6 features with 40% or more missing values. Finally, we designated the remaining 67 features as either numerical or categorical features. Apart from that, we also tried performing standard scaling and outlier removal using Z-score technique. But scaling and outlier removal didn't improve the overall model accuracy. So, we didn't include them in our final pipeline.

6.2 Multi-model Implementation and Model Selection

In Table 1, we summarized all the model used in this study. We have a huge dataset, and because of source and time constraints, we used a specific sample size to train certain models. We also reported the accuracy of those models and their weakness and strength to certain parameters such as runtime, accuracy, memory efficiency. The most prominent weakness of the models is the accuracy and runtime. We have some tradeoff between runtime and accuracy. If we wanted to get higher accuracy in certain models, we need to use the whole dataset to train. Unfortunately, it put constraints on both resource (RAM) and time. For that reason, we just used a small fraction of the dataset. Our final goal is to achieve a higher accuracy using a SuperLearner. We have to consider resource and time constraints to train the models. Considering these factors, we selected five baseline models from the boosting family to build the Super-learner.

6.3 Feature Selection

In Figure 9, 10, 11, 12 We generated and showed feature importance based on the decision tree,LGB, XGB, and random forest classifier. The reason for feature selection is to select the most critical features and show their effects on the model's accuracy. Here, the bar plots show a sorted list of features based on their importance score for each model. We ranked the top features combining the decision tree, LGBM, XGB, and random forest classifier. To see the effects of the feature on the performance, we gradually decrease the number of features and train two top models from Table 1 LGBM, and Catboost classifier and summarized the results on Table 2. From Table 2, the best performance is achieved when we used the top 57 features set to train the models. The top 57 feature set outperforms all other features combination. For this reason, we use those selected 57 features to train the SuperLearner in the next step.

Model	Sample Size (%)	Accuracy	Weakness	Strength
Gaussian Naïve Bayes	100%	0.5004	Most features categorical	Runtime
Multimodal Naïve Bayes	1%	0.5048	Continuous features, accuracy	-
Support Vector Machine	1%	0.5312	Runtime, accuracy	-
Multi-layer Perceptron Classifier	5%	0.5143	Runtime, accuracy	-
Gaussian Mixture	1%	0.52	Accuracy	Runtime
Logistic Regression	1%	0.63	-	Runtime, accuracy
Ridge Classifier	1%	0.62	-	Runtime, accuracy
Random Forest Classifier	100%	0.65	not memory efficient	Runtime, Skewness
KNN	1%	0.61	Runtime	Accuracy
Gradient Boosting Classifier	**100%**	**0.66503**	-	**Accuracy, Runtime**
LGBM Classifier	**100%**	**0.67191**	-	**Accuracy, Runtime**
XGB Classifier	**100%**	**0.65593**	**Accuracy**	-
Cat Boost Classifier	**100%**	**0.67563**	-	**Accuracy, Runtime**
AdaBoost Classifier	**100%**	**0.65041**	**Accuracy**	**Accuracy, Runtime**

Table 1: Different models, percentage of training data to train, test accuracy, models' weakness, and strength

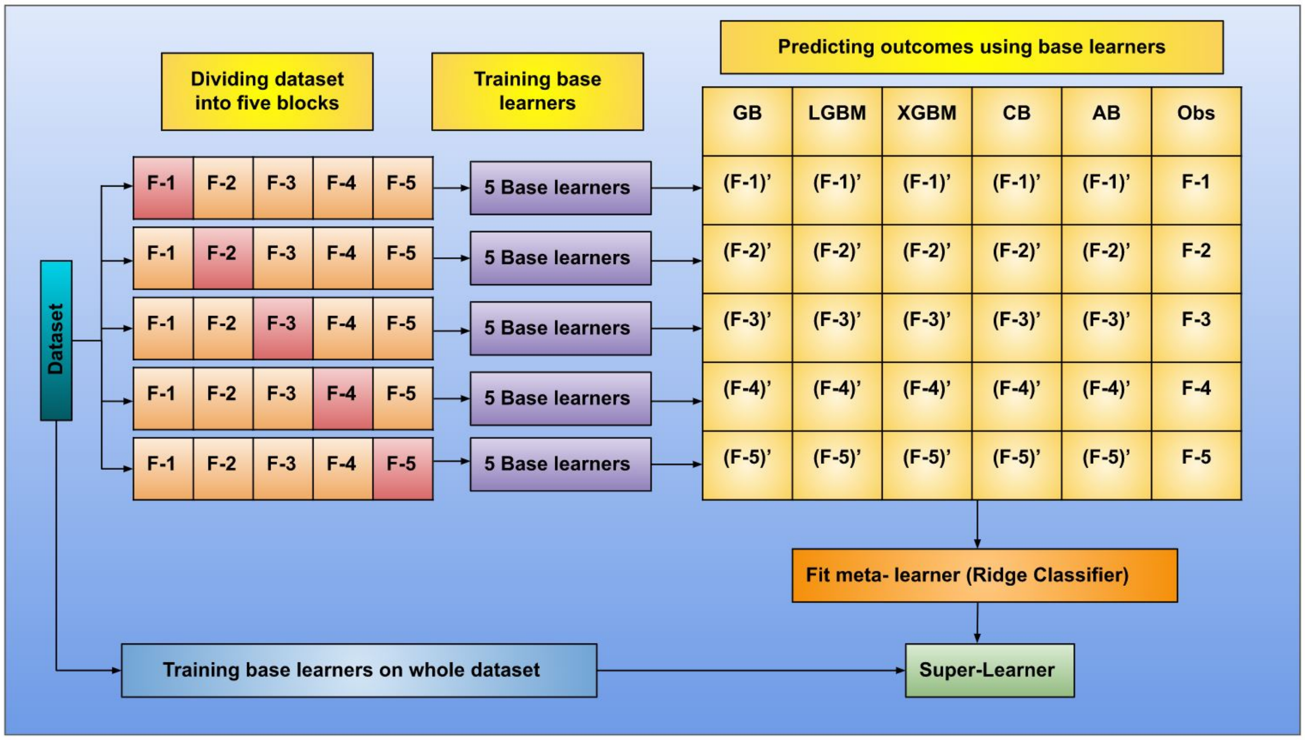
Dividing dataset into five blocks
Training base learners
Predicting outcomes using base learners
GB
LGBM
XGBM
CB
AB
Obs
Dataset
F-1
F-2
F-3
F-4
F-5
5 Base learners
(F-1)'
(F-2)'
(F-3)'
(F-4)'
(F-5)'
Fit meta- learner (Ridge Classifier)
Training base learners on whole dataset
Super-Learner

Figure 8: SuperLearner

6.4 Super-Learner

In Figure 8, we showed the proposed workflow of the Super-Learner. We split the whole dataset to 80-20%. 80% data was used to train the model, and the other 20% was used for testing purposes. To build the Super-Learner, we first divided the training dataset into five non-overlapping blocks (F-1, F-2, F-3, F-4, F-5).In the first row, we used the F-1 block for cross-validation. Specifically, we used blocks (F-2, F-3, F-4, F-5) to train five base learners and use F-1 to make the prediction using those trained models. We got the first row of the predicting outcomes using base learners block using this workflow. We repeat the same procedures for the rest of the blocks. In this way, we build feature sets for meta learners. For meta-learner, we use ridge classifier as it has less time to train and also worked optimally compared to other meta-models, .i.e., logistic regression, GBM. On the other hand, we used the whole dataset to train all the base learners. Then, we combined the meta-learner and base models to create Super-Learner. When testing the Super-Learner on a test dataset, we first get the base learners' output and stack them horizontally. That stacked output then passed through the meta-learner to get the final prediction from the

Feature Set	Model	Accuracy	Precision	Recall	F1-Score
67 Feature set	LGBM Classifier	0.67133	0.6731	0.6713	0.6713
	Cat Boost Classifier	0.67502	0.6765	0.675	0.6751
Top 57 Feature set	LGBM Classifier	**0.67193**	0.6736	**0.6719**	**0.6719**
	Cat Boost Classifier	**0.67563**	**0.6771**	**0.6756**	**0.6757**
Top 52 Feature set	LGBM Classifier	0.67154	0.6734	0.6715	0.6715
	Cat Boost Classifier	0.67524	0.6771	0.6752	0.6752
Top 47 Feature set	LGBM Classifier	0.67184	**0.6737**	0.6718	0.6718
	Cat Boost Classifier	0.67524	0.677	0.6752	0.6752
Top 42 Feature set	LGBM Classifier	0.66984	0.6714	0.6698	0.6698
	Cat Boost Classifier	0.67249	0.6736	0.6725	0.6726
Top 37 Feature set	LGBM Classifier	0.67081	0.6727	0.6708	0.6708
	Cat Boost Classifier	0.67371	0.6754	0.6737	0.6737
Top 31 Feature set	LGBM Classifier	0.66952	0.6711	0.6695	0.6693
	Cat Boost Classifier	0.67202	0.6734	0.672	0.6719
Top 21 Feature set	LGBM Classifier	0.66344	0.6651	0.6634	0.6633
	Cat Boost Classifier	0.66625	0.6677	0.6663	0.6661
Top 16 Feature set	LGBM Classifier	0.65956	0.661	0.6596	0.6593
	Cat Boost Classifier	0.66146	0.6628	0.6615	0.6613

Table 2: Different number of features set used to see the performance of the top two model with respect to accuracy

Super-Learner.

Base Learners	Model	Accuracy	Precision	Recall	F1-Score
Indiviual Base Learners Results (using 57 feature set)	Gradient Boosting Classifier	0.66503	0.6659	0.665	0.6651
	LGBM Classifier	0.67191	0.6736	0.6719	0.6719
	XGB Classifier	0.65593	0.6582	0.6559	0.6558
	CatBoosting Classifier	0.67563	0.6771	0.6756	0.6757
	AdaBoosting Classifier	0.65041	0.651	0.6504	0.6505
5 Base Learners (GB,LGBM,XGBM,CB,AB)	SuperLearner	0.67585	0.677	0.6759	0.6759
4 Base Learners (GB,LGBM,XGBM,CB)	**SuperLearner**	**0.67592**	0.677	**0.6759**	**0.676**
3 Base Learners (GB, LGBM, CB)	SuperLearner	0.67566	**0.6772**	0.6757	0.6757
2 Base Learners (LGBM,CB)	SuperLearner	0.67559	0.677	0.6756	0.6756

Table 3: Results of Individual base learners and Super-Learner using different combination of base learners

7 Results

For evaluating the results of Super-Learner, we use a different combination of base learners according to their performance. At first, we use all five base learners to create the Super-Learner. Table 3 shows the accuracy, precision, recall, and f1 score of the model on test data. We have slight improvement on the accuracy of Super-Learner comparing to the best performing base model Catboost classifier. Then, we dropped low-performing base learners Adaboosting to see the performance. The accuracy of the Super-Learner further improves with dropping the Adaboosting classifier. When we drop the XGB classifier from the Super-Learner, the accuracy drops, although its performance is slightly better than the Catboosting classifier. When we dropped the Gradient Boosting classifier, the performance of the Super-Learner further deteriorates, and the accuracy is less than the Catboost classifier. From these results, we propose that using four base learners GB, LGBM, XGBM, and CB, is optimal for building Super-Learner for the underlying problem. Our proposed model has an accuracy of 67.6% while having a precision of 67.7%, 67.6% recall value and 67.6% f1-score. In order to analyze the result better, Figure 13 shows the confusion matrix for the proposed model. As we can see in the figure, both the probability of predicting 'Has detected Malware' and 'Has not detected Malware' is more than 65%. Apart from that, the receiver operating characteristic (ROC) curve shown in Figure 14 shows that the output is more aligned towards True Positive Rate (TPR) than compared to False Positive Rate (FPR) with an AUC score of 0.6765.

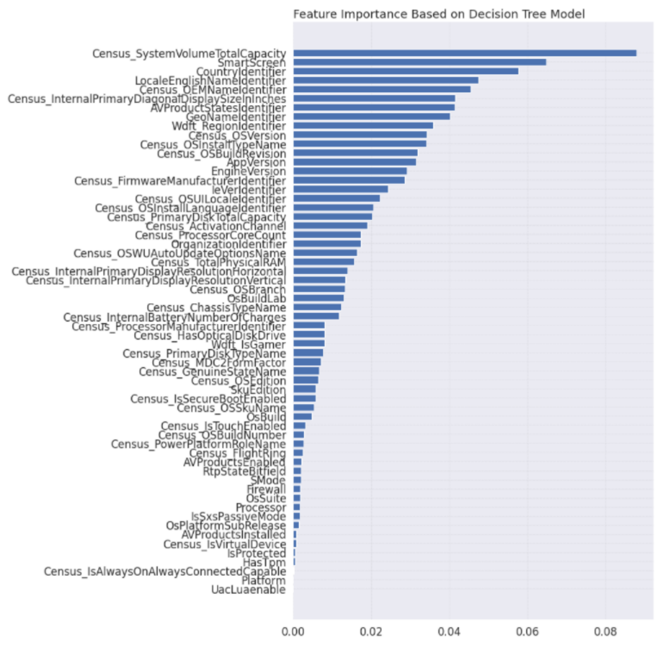

Figure 9: Feature Selection based on Decision Tree model

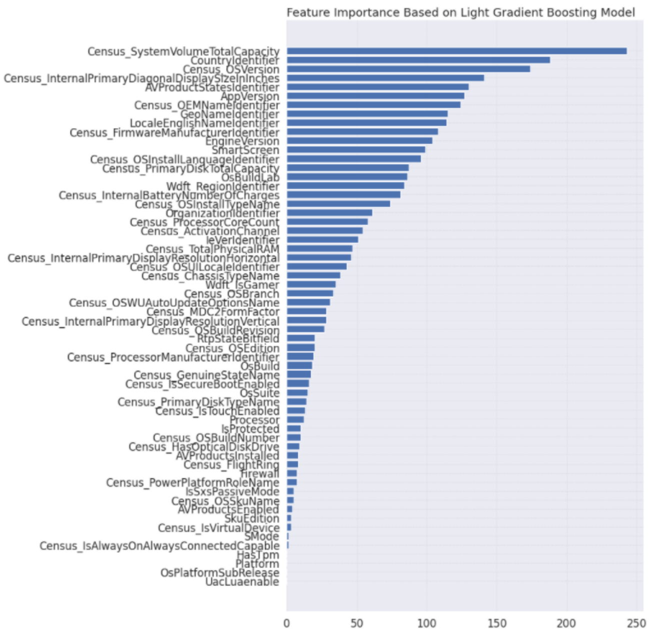

Figure 10: Feature Selection based on LGBM model

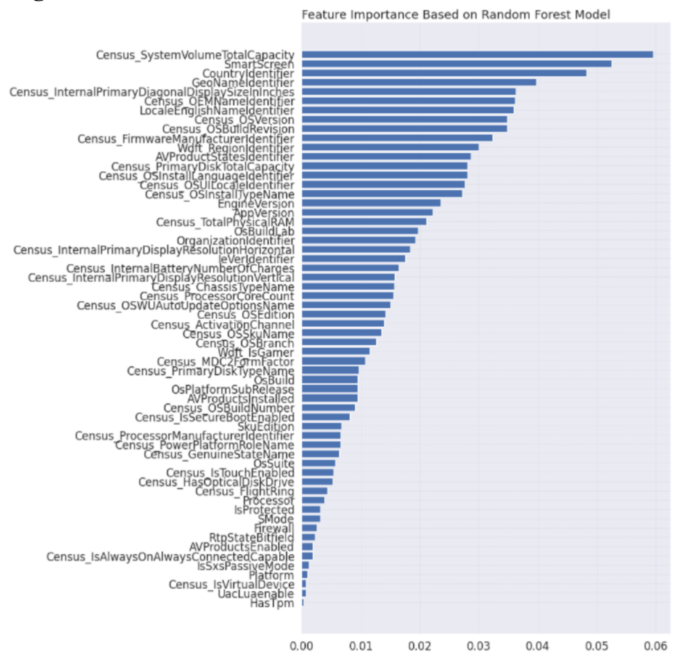

Figure 11: Feature Selection based on Random Forest model

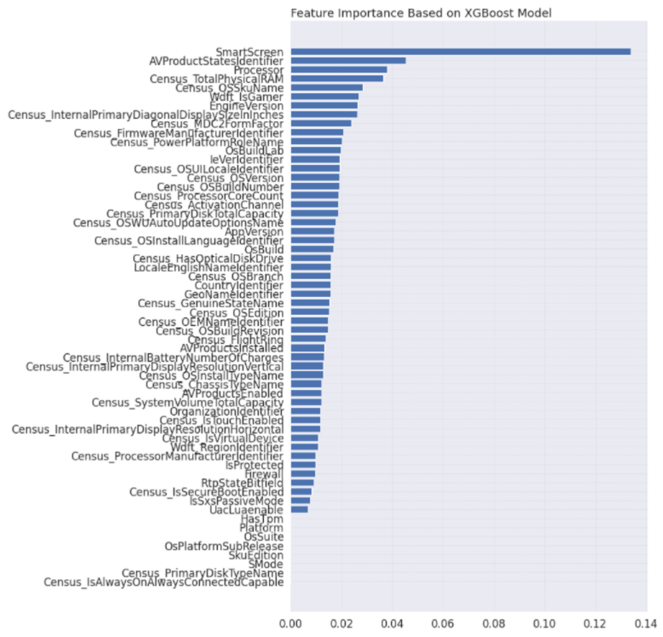

Figure 12: Feature Selection based on XGBM model

8 Related Work

Due to the proliferation of applying machine learning to time series data problems across different research domains, our team chose to examine works based on implemented Super-Learner machine learning models. Two works, one in the healthcare domain [7] and another one in the natural science domain [9], implemented Super-Learner machine learning models to address research problems in their respective domains. [7] utilized a Super-Learner machine learning model which relies on V-fold cross-validation to predict the propensity score, i.e., conditional probability, of treatment assignments given baseline covariates by comparing candidate machine learning algorithms. Conversely, [9] implemented a Super-Learner machine learning model comprised of various base learner algorithms, such as linear regression, random forests, polyMARS, MARS, Lasso, and support vector regression. The Super-Learner the authors implemented in [9] was designed to forecast daily

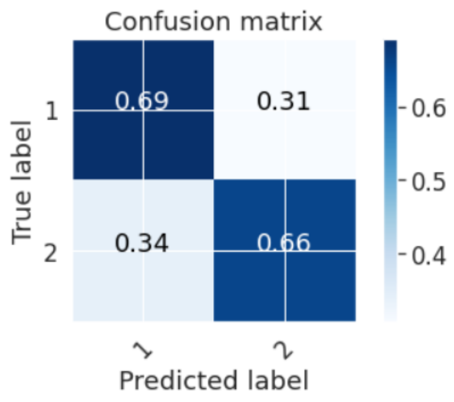

Figure 13: Confusion Matrix Generated from the Result of Super Learner

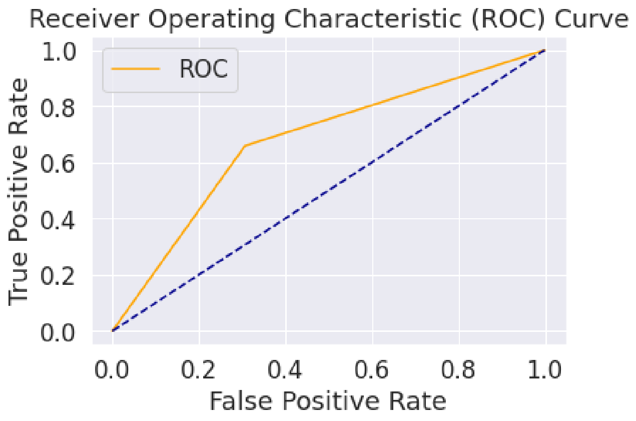

Figure 14: ROC Curve Generated from the Result of Super Learner

streamflow.

9 Discussions

The dataset is really large in size and a high number of dimensionality and cardinality with lots of missing values. It also doesn't show any significant correspondence among the features and contains a lot of noise. All these features make this dataset a bit difficult to work with. Classic binary classifiers like Naive Bayes, SVM, Multi-layer Perceptron, Gaussian Mixture models requires huge resource and time to train on this dataset and still the accuracy is pretty poor outside the context of our problem. Algorithms like KNN, Random forest, Logistic Regression and Ridge Classifier did perform slightly better but still their runtime is long and accuracy isn't good enough outside the context of our problem. On the other hand, Boosting algorithms like Gradient Boosting, LGBM Classifier, XGB Classifier, Cat Boost Classifier and AdaBoost Classifier performed comparatively well in shorter runtime. The reason behind this can be single classifiers are working as weak learner here with accuracy slightly above random choice, so boosting is improving the overall accuracy of the weak learners. Finally, when we implemented a super learner on top of four boosting algorithms, the accuracy improved even further. However, the current obtained accuracy is still slightly less than 70%, but still in the upper positions of the competition's leaderboard, and further improvement might be possible on this model by handling data noise and outliers in more effective ways.

10 Conclusion

As the era is bending more towards scalable technologies and cloud computing, cybersecurity is of ever increasing concern. Having a tool that can predict the possibility of a device being vulnerable to certain malware will not only decrease the overall risk, it will also give us better insight into features that make devices less vulnerable to threats. This can be invaluable to help advance modern technologies and cybersecurity. However, this is not an easy problem to solve. In the context of our problem, our proposed Super-Learner model showed an improvement in accuracy over the accuracy of individual base learners. Therefore, this result makes our work promising for further work and exploration.

An implementation for our project is available at the following link: https://github.com/NabilaKhan/CAP-5610-Machine-Learning-/blob/main/CAP_5610_Final_Project_SuperLearner.ipynb

References

[1] AV-test. https://www.av-test.org/en/statistics/malware/. Accessed: 2021-04-29.

[2] AV-test. https://www.kaggle.com/chandrimad31/malware-prediction-using-dask-for-handling-bigdata. Accessed: 2021-04-29.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the Twenty-second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, August 2016.

[4] A. V. Dorogush, V. Ershov, and A. Gulin. Catboost: gradient boosting with categorical features support. *Proceedings of ML Systems Workshop*, December 2017.

[5] Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, pages 771–780, September 1999.

[6] J. H. Friedman. Greedy function approximation: A gradient boosting machine. Technical report, Sequoia Hall, Stanford University, Stanford, CA 94305.

[7] C. Ju, M. Combs, S. D. Lendle, J. M. Franklin, R. Wyss, S. Schneeweiss, and M. J. van der Laan. Propensity score prediction for electronic healthcare databases using super learner and high-dimensional propensity score methods. *Journal of Applied Statistics*, pages 2216–2236, February 2019.

[8] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Proceedings of the Thirty-first Conference on Neural Information Processing Systems*, pages 3149–3157, December 2017.

[9] H. Tyralis, G. Papacharalampous, and A. Langousis. Super ensemble learning for daily streamflow forecasting: Large-scale demonstration and comparison with multiple machine learning algorithms. *Neural Computing and Applications*, pages 3053–3068, July 2020.