Microsoft Malware Prediction

Bhavna Arora
Dept. of CSE
PES University
Bengaluru, India
bhavnaaro28@gmail.com

Akanksha Tonne
Dept. of CSE
PES University
Bengaluru, India
tonne.akanksha99@gmail.com

Athira AD
Dept. of CSE
PES University
Bengaluru, India
athiraasha1126@gmail.com

***Abstract*— In this paper we have presented a method to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. The dataset used for the classification is not a representative of Microsoft customer's machines in the wild; it has been sampled to include much larger proportion of malware machines. Various preprocessing methods and dimensionality reduction techniques are used for getting accurate results of classification. Light GBM (Gradient Boosting), a tree-based classifier is used for obtaining the important features from our dataset. Using these features along with the external information provided by the kaggle we built machine learning models for binary classification.**

Keywords— Malware Prediction, Microsoft Malware, Computer and Network security, Potential threats

I. INTRODUCTION

Malware is a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network. It is designed to bypass security systems and avoid detection, making it extremely difficult for security teams to ensure that users and the wider business are not adversely impacted. Measured in terms of worldwide user numbers, Windows remains the number one operating system. The players in the malware industry are in full agreement, and so Microsoft systems are still cybercriminals main target of attack.

The prediction on malware behavior or development is as crucial as the removing of malware itself. This is because the prediction on malware provides information about the rate of development of malicious programs in which it will give the system administrators prior knowledge on the vulnerabilities of their system or network and help them to determine the types of malicious programs that are most likely to taint their system or network.

This project uses the Microsoft Malware Dataset released during the 2015 Malware Challenge, available on Kaggle.

II. PREVIOUS IMPLEMENATIONS

The problem statement being analyzed in this paper had been a Kaggle competition, hence a lot of implementations were available online which helped in gaining some of the not so obvious insights from the data. The link [4] helped us to understand that converting the datatype of the attributes to a lower range can help in loading the data faster. The link [5] was referred to understand and incorporate the externally.

III. PROBLEM STATEMENT

The goal of this competition is 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.

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. The dataset provided here has been roughly split by time.

IV. APPROACH

A. Understanding the dataset

The size of the training and testing data is 9 million and 8 million rows, respectively. There are 81 features in total, with 52 being categorical, 23 of which are encoded numerically to protect the privacy of the information.

On analyzing the dataset we realized a few challenges faced are listed as follows:

- Large Dataset
- Many attributes
- Missing Values
- Categorical features

It is essential to resolve each of the above issues in the preprocessing stage before moving forward with the any visualization and classification.

Table 1: Snapshot of the dataset

	IsBeta	RtpStateBitfield	IsSxsPassiveMode	DefaultBrowsersIdentifier	AVProductStatesIdentifier	AVProductsInstalled	AVProductsEnabled	HasTpm
count	8.921483e+06	8889165.0	8.921483e+06	433438.000000	8.885262e+06	8885262.0	8885262.0	8.921483e+06
mean	7.509962e-06	NaN	1.733378e-02	1658.903809	4.948320e+04	NaN	NaN	9.879711e-01
std	2.740421e-03	0.0	1.305118e-01	999.028870	1.379994e+04	0.0	0.0	1.090149e-01
min	0.000000e+00	0.0	0.000000e+00	1.000000	3.000000e+00	0.0	0.0	0.000000e+00
25%	0.000000e+00	7.0	0.000000e+00	788.000000	4.948000e+04	1.0	1.0	1.000000e+00
50%	0.000000e+00	7.0	0.000000e+00	1632.000000	5.344700e+04	1.0	1.0	1.000000e+00
75%	0.000000e+00	7.0	0.000000e+00	2373.000000	5.344700e+04	2.0	1.0	1.000000e+00
max	1.000000e+00	35.0	1.000000e+00	3213.000000	7.050700e+04	7.0	5.0	1.000000e+00

B. *Preprocessing*

- Large Dataset:

  The dataset is huge with datatypes of the columns occupying significant amount of memory. To combat this problem, the columns' datatypes are converted to a lower datatype reducing the memory occupied with decrease in data load time.

  Eg: a column with datatype int16 is converted to int8

- Missing Values

  If not dealt appropriately, the missing data could lead to wrong inference from the data.

  1. **Missing value ratio**: Columns with more than 50% missing values are dropped from the data frame. There were 7 such columns.
  2. Impute the missing values: Since, most the data is categorical, the rows having missing values cannot be replaced mean or median.

  The solution we used for this problem was to replace the missing categories with the most occurring category in that column i.e. with the mode the column.

- Categorical Features:

  A variable having numeric type does not imply that it is quantitative, the numbers could represent categories/levels also.

Low variance filter: From inspection of the dataset (nature and domain of the values an attribute can take) and calculating number of unique values under each columns, if a column contains categorical data but 80% of the rows fall in the same attribute v, then that attribute would not play much significance in prediction. 26 columns are dropped from the dataset using this concept.

C. *Data Preparation*

LGBM model helped us to estimate the important features in our dataset. From the Figure 1 we can understand that out of 81 features the most important features for the classification are AvSigversion, Census_OS_Version, and OsBuildLab.

- AvSigversion:- Defender state information e.g. 1.217.1014.0
- Census_OS_Version:- Numeric OS version Example - 10.0.10130.0
- OsBuildLab: Build lab that generated the current OS.

Example :9600.17630.amd64 fre.winblue_r7.150109-2022

On further analysis of dataset we could understand that version numbers in the Microsoft malware data are associated with time. Microsoft publishes time stamps for AvSigVersion and Census_OSVersion. Additionally we can deduce time stamps for OsBuildLab and EngineVersion. The data consists of mapping between the AvSigversion and the date. This mapping is used to replace AvSigversion with its corresponding date.

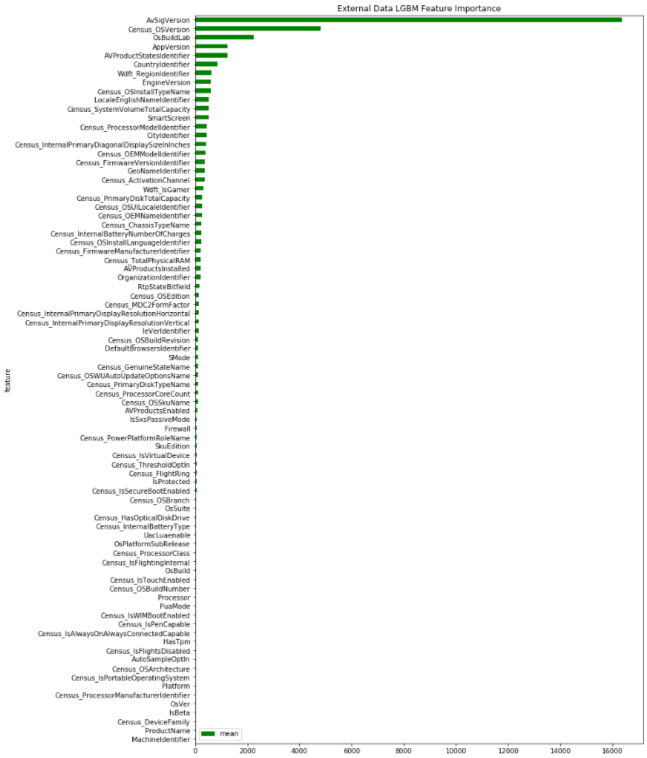


Fig1: Feature Importance from original dataset\

The mapped date is used to merge with the Google Data.

These columns are mapped to Google Data.

Google Data

Also Google publishes information about malware on the internet over time. Some of the entries in this data include

1. Malware sites detected per week
2. Phishing sites detected per week
3. Attack sites(These are websites that hackers have set up to intentionally host and distribute malicious software)
4. Compromised sites (These are legitimate websites that have been hacked to include content from, or to direct users to, sites that may exploit their browsers)

Threat Data

Microsoft also publishes malware threats. For each AvSigVersion, this external dataset lists all the known malware that was threatening it. Eventhough there is a lot of information on this dataset we included only threatening count for each AvSigVersion.

The final dataset is a combination of Google Data and Threat Data time mapped with AvSigversion, Census_OS_Version, and OsBuildLab. Final dataset is split into train and test in the ratio 7:3.

Feature importance graph, Figure 3, for the merged data shows that all the features considered are significant.

Prediction in the combined data comes into the class of time series classification.

D. *Visualizations*

Since the original dataset was mostly categorical not many visualizations were possible. Mapping the AvSigVersion of the original dataset with the timestamps from another external data, we were able to get more insights from the data.
Most of the detections were found to be in the months of July, August and September.

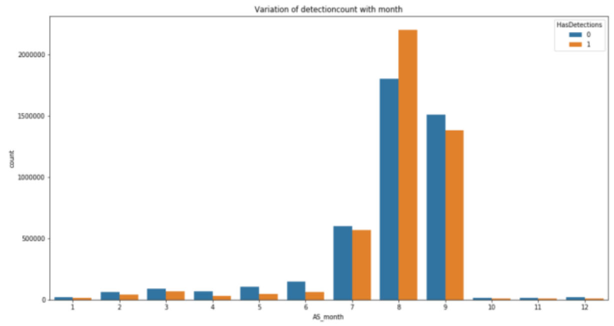


Fig2. Monthwise variation of detection count

From the graph we can say that systems with smartscreen requiring admin persmission are less likely to be infected with malware.

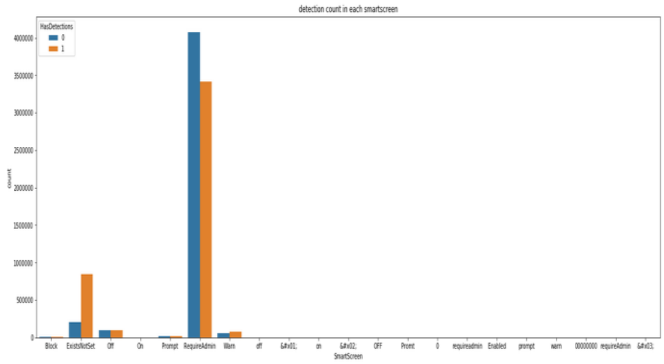


Fig3. relation between SmartScreen and hasDetections

E. *Model Fitting*

1) *Light GBM*

Light GBM is a gradient boosting framework that uses tree based learning algorithm.
Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.
In our context the reason for choosing Light GBM were:

- Large dataset
- Too many categorical variables

2) *Long Short Term Memory Units (LSTM)*

Most common way of time series classification is using LSTMs.

LSTM can store information about previous values and exploit the time dependencies between the samples. Adam optimizer with a learning rate of 0.001 was used to define the model's error minimization approach.

For classification of the pre-processed data, experimentation with varying parameters like number of LSTM units and hidden layers is done to find the optimal parameters. For various parameters, Figure 2, accuracy does not deviate significantly. The scale transformation of data did not affect the accuracy significantly.

3) *Adaboost*

Adaboost is an ensembling learning technique. It combines multiple weak learners to produce a final strong classifier. The accuracy obtained is 55%. There is no significant change in accuracy for varied number of learners. In our project we used Adaboost model with 25 weak learners for classification.

V. Evaluation Metric

For light GBM we use AUC as the metric for evaluation.
In case of LSTM and Adaboost, accuracy is chosen as an evaluation metric.

VI. Experimentand Results

1. Light GBM was used for classification and AUC of 0.72 was obtained. Since light GBM is a tree based algorithm, important features are found using the feature importance attribute graph.
2. From the feature importance graph only 3 features (AVSigVersion,Census_OS_build, OS_Build_Lab) play the major role in predicting if a system would be affected with malware.
3. On further exploration it was found that there are additional datasets available which can be merge with the original dataset to give more details about the 3 most important features found.
4. The combined dataset consists of only numerical columns. Hence, Adaboost is used as a classifier.
5. Since the dataset had time indicative components, which Adaboost doesn't address, LSTM was used. LSTM can be used when we need to perform classification on time dependent data. It resulted in a poor performance.

The following table summarises the accuracy for various model parameters:

Table 2. LSTM accuracies with varying parameters.

Number of LSTM units	Activation Function	Accuracy(%)
100	Sigmoid	49.97
200	Sigmoid	49.99
500	Sigmoid	49.98
100	ReLu	50.00
200	ReLu	50.00
500	ReLu	50.00

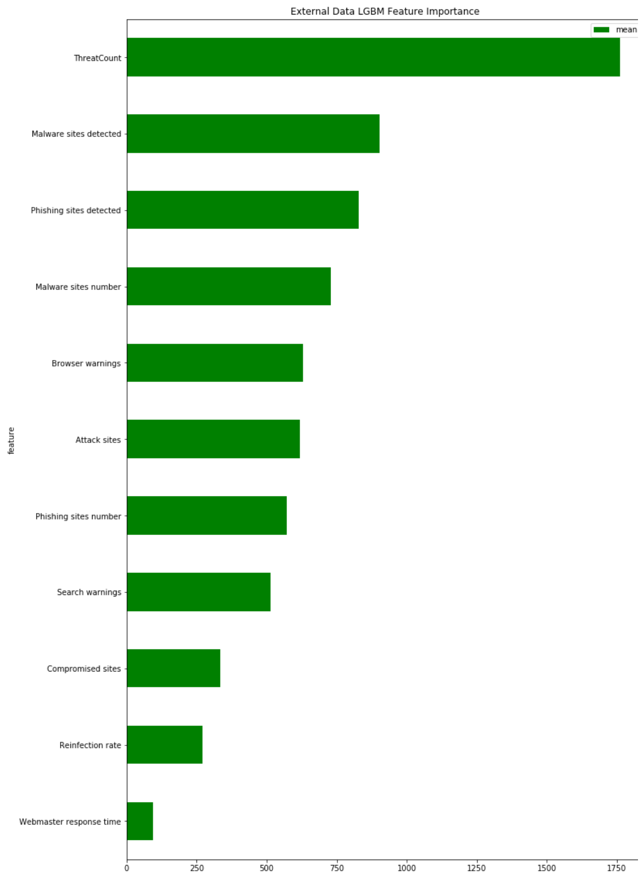
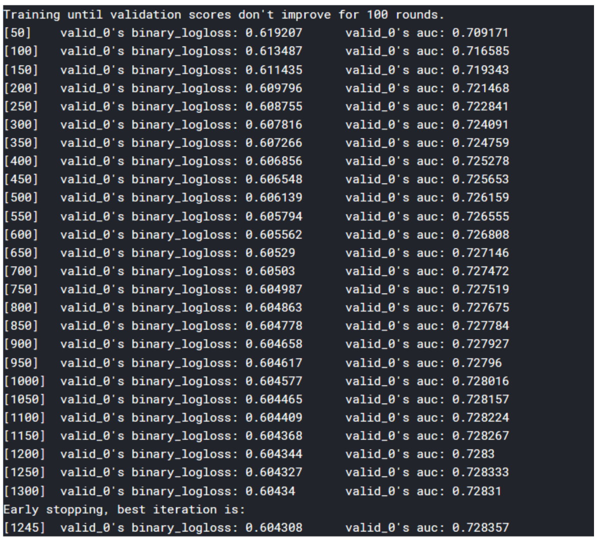


Fig4. Feature Importance from merged dataset



```
Training until validation scores don't improve for 100 rounds.
[50]    valid_0's binary_logloss: 0.619207      valid_0's auc: 0.709171
[100]   valid_0's binary_logloss: 0.613487      valid_0's auc: 0.716585
[150]   valid_0's binary_logloss: 0.611435      valid_0's auc: 0.719343
[200]   valid_0's binary_logloss: 0.609796      valid_0's auc: 0.721468
[250]   valid_0's binary_logloss: 0.608755      valid_0's auc: 0.722841
[300]   valid_0's binary_logloss: 0.607816      valid_0's auc: 0.724091
[350]   valid_0's binary_logloss: 0.607266      valid_0's auc: 0.724759
[400]   valid_0's binary_logloss: 0.606856      valid_0's auc: 0.725278
[450]   valid_0's binary_logloss: 0.606548      valid_0's auc: 0.725653
[500]   valid_0's binary_logloss: 0.606139      valid_0's auc: 0.726159
[550]   valid_0's binary_logloss: 0.605794      valid_0's auc: 0.726555
[600]   valid_0's binary_logloss: 0.605562      valid_0's auc: 0.726808
[650]   valid_0's binary_logloss: 0.60529       valid_0's auc: 0.727146
[700]   valid_0's binary_logloss: 0.60503       valid_0's auc: 0.727472
[750]   valid_0's binary_logloss: 0.604987      valid_0's auc: 0.727519
[800]   valid_0's binary_logloss: 0.604863      valid_0's auc: 0.727675
[850]   valid_0's binary_logloss: 0.604778      valid_0's auc: 0.727784
[900]   valid_0's binary_logloss: 0.604658      valid_0's auc: 0.727927
[950]   valid_0's binary_logloss: 0.604617      valid_0's auc: 0.72796
[1000]  valid_0's binary_logloss: 0.604577      valid_0's auc: 0.728016
[1050]  valid_0's binary_logloss: 0.604465      valid_0's auc: 0.728157
[1100]  valid_0's binary_logloss: 0.604409      valid_0's auc: 0.728224
[1150]  valid_0's binary_logloss: 0.604368      valid_0's auc: 0.728267
[1200]  valid_0's binary_logloss: 0.604344      valid_0's auc: 0.7283
[1250]  valid_0's binary_logloss: 0.604327      valid_0's auc: 0.728333
[1300]  valid_0's binary_logloss: 0.60434       valid_0's auc: 0.72831
Early stopping, best iteration is:
[1245]  valid_0's binary_logloss: 0.604308      valid_0's auc: 0.728357
```

Fig5. Obtained results from Light GBM on combined data

VII. Conclusion and Insights

Adaboost Classifier and LSTM do not give a good accuracy for combined data. Changing parameters do not affect the results. Low accuracy indicates that the merging the datasets may not have captured the essence of the actual classification problem.

Light GBM does not require categorical features to be encoded and for a time dependent classification it suits well. It is fast when applied to large datasets.

VIII. References

[1] Microsoft Malware Challenge, https://arxiv.org/abs/1802.10135

[2] A Hands-On Introduction to Time Series Classification (with Python Code) ;Available at: https://www.analyticsvidhya.com/blog/2019/01/introduction-time-series-classification/

[3] https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

[4] https://www.kaggle.com/theoviel/load-the-totality-of-the-data

[5] https://www.kaggle.com/cdeotte/external-data-malware-0-50

[6] Quick Introduction to Boosting Algorithms in Machine Learning https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/