# Predicting the Vulnerability of Windows Machines to Malware

Ayan Majumdar
2571656

Anilkumar E S
2571210

Barno Kaharova
2528389

## Abstract

*The goal of this project is to use Machine Learning models to predict the probability of a Windows machine being vulnerable to a malware attack based on features describing the machine configurations. It is based on the open source Kaggle competition. We perform data analysis, design and compare 5 different algorithms for the task and do feature ablation study. Our analysis shows that it is indeed possible to predict the vulnerability of Windows systems provided the features, and also how some models perform better in the task than others.*

## 1. Introduction

The malware industry continues to be a well-organized, well-funded market dedicated to evading traditional security measures. Once a computer is infected by malware, criminals can hurt consumers and enterprises in many ways. Thus this problem should be addressed to improve the security.

Machine Learning and advanced Data Analysis techniques have already been used in the domain of Malware Analysis, like the previous Microsoft Malware Classification Challenge (BIG 2015) on Kaggle. The idea of this project is to see if we can apply such algorithms to detect the vulnerability of any system to malware. Given a dataset that consists of specification of several Windows systems, the goal is to predict which machine can be targeted by malware in the future given the features provided. To the best of our knowledge, there has not been much work that has been done on this front, for example [1] uses Bayesian State Space Model to forecast malware detection of cyber events. However, using such large scale data as our project to predict malware vulnerability from system specifications has not been addressed enough.

## 2. Methodologies

We work with a subset of the data provided by Microsoft as part of the Kaggle Competition for our experiments and analysis. Overall, our work involved data partition, exploratory data analysis, data pre-processing, fitting Machine Learning models and evaluating and comparing the performance. We now shortly describe each of the steps.

### 2.1. Dataset and Data Partition

The original dataset [9] contains almost 9 million data entries. For our analysis, we decided to work on a subset of the dataset. From the `train.csv` file provided, we select 1 million rows for our training data and 250k rows for our test data. As the original `test.csv` did not have the true labels, it was not possible to take data from this and perform analysis for us, hence we decided to take data from the training file alone.

### 2.2. Exploratory Data Analysis

Next, we perform exploratory data analysis for multiple features provided. This allows us to get a good idea of what each feature corresponds to and what our data looks like. The data includes multiple features with descriptions that are available at [9].

We analyze the target variable distribution and see that our data is balanced. Next, we try to see which features have most of the missing data and we would go on to drop the top 5 of the features which have more than 70% data missing. For the visualizations kindly refer our notebook.

We perform EDA on selected categorical features. Here we show the result for a feature named Smart Screen (Figure 1). We also measure similarity among features by correlation, e.g. we find `OsPlatformSubRelease` and `OsBuild` are highly related. For the full analysis, kindly refer to the notebook.

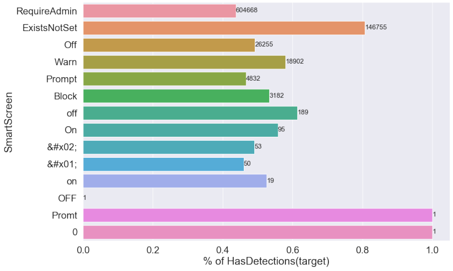

Figure 1: Smart Screen Feature EDA (Detection Rate)

## 2.3. Data Pre-processing

We encode the categorical features for downstream tasks. We analyze which categorical features have high cardinality and based on that we perform frequency encoding on them. For the others, we perform normal label encoding. More information about this can be found in [11]. The basic idea is to encode categorical features with high cardinality with the frequency of occurrence of each level, so that higher frequency levels are given more importance. We also impute the missing values by replacing with median to ensure that it is not affected by outliers.

## 2.4. Evaluation Criteria

Our primary task is binary classification, so we choose Receiver Operating Characteristic Area Under curve (ROC AUC) [15] along with accuracy as our metrics.

# 3. Models

Once we have prepared our data, the next step is to select suitable models to fit the data and analyze the predicting capability of them. As part of this project we experiment with 5 different models, 4 of them being standard machine learning models and one deep learning model. Note that we first fit all the models on the full set of features minus the top 5 features that have most missing data. Later we also perform a feature ablation study.

## 3.1. Machine learning Models

Standard machine learning algorithms have become quite popular and are also extremely effective in multitude of applications. Going with this, we decide to test four different models on the data that range in complexity. We discuss the models we used below.

### 3.1.1 Logistic Regression (Baseline)

This is the simplest model and our baseline. We try to see if there is a linear relationship between the features and the target. We use L2 regularization of the weights.

### 3.1.2 Decision Tree

As our next model we fit a Decision Tree. We try to vary the number of samples at the leaves (1, 20, 40) to improve performance. This feature is important to improve generalization and prevent overfitting.

### 3.1.3 Random Forest

Next, we wanted to extend to ensemble techniques, specifically Random Forest Classifier [14]. We varied the hyperparameter number of estimators/trees [2] (10, 25, 50) to improve performance. For the parameter number of samples at leaves we keep it the same as Decision Tree (40). Random Forests use a fraction of features at each split and we set this to the suggested `sqrt(num_features)`.

### 3.1.4 Light Gradient Boosting Method (LGBM)

Gradient boosting is a boosting algorithm in which the loss is minimized using Gradient Descent method. Light GBM [4], [8] is a fast, distributed, high-performance gradient boosting framework. Unlike other boosting algorithms it splits the trees leaf wise and not level wise. LGBM runs very fast, hence the word 'light'. LGBM has a vast set of hyperparameters that can be tuned to get the optimal performance [6].

For this problem we have selected two of the most important hyperparameters (`num_leaves` and `min_data_leaf`) to build the model. Overall we built 10 different models to analyze the effect of `num_leaves` and `min_data_leaf` on the model performance. Parameter `num_leaves` represents the number of leaves in a full tree and this controls the complexity of the tree model which in turn can be used to regulate the overfitting. Parameter `min_data_leaf` is the minimum number of samples required in a leaf node [7].

LGBM classification model use sigmoid as activation function, L1 regularization as regularizer, and cross entropy as the loss function. Designed model uses Gradient Boosted trees as boosting method which uses Decision Trees and Stochastic gradient descent for boosting. Performance metric of LGBM model is AUC-ROC and accuracy. Figure 2 depicts how the model performance(accuracy) varies with hyperparameters `num_leaves` and for the analysis of hyperparameter tuning of `min_data_leaf` we refer to our notebook.

## 3.2. Neural Network Models

Given the scale of the data it also made sense to try to build a deep learning model to predict from the data. We try to build feed-forward models. As per [12] we include Embedding layers for categorical features. In addition we scale the data [10] before feeding it to the neural network by using a `MinMaxScaler`. We vary the number of layers (simplest model is 4 layers, Embedding-128-64-32-Sigmoid. Each model with extra layers has double the neurons, for example one extra layer means Embedding-256-128-64-32-Sigmoid.), try Dropout (at dropout rate of 0.2, embedding dropout at 0.3), Batch Normalization [3]. We use binary cross entropy as the loss function, ReLU activation at all layers except the last (sigmoid), Adam Optimizer (default learning rate of 0.001) [5], batch size of 512 and train for 8 epochs. Overall, we develop 9 models, for more details please refer to our notebook.

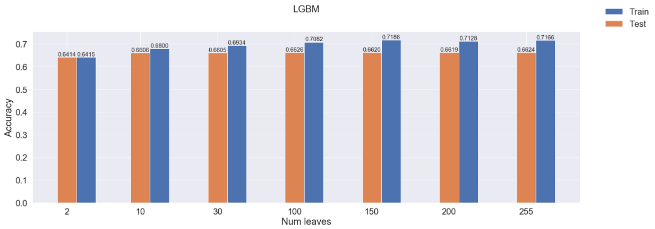

Figure 2: LGBM: Num Leaves vs Accuracy

# 4. Results and Observations

Now we discuss and compare the outputs of the models after we fit them and used them to predict on the test data. For the first three models (Logistic Regression, Decision Tree, Random Forest) we use 5-fold cross validation and for LGBM and Neural Network models we split the training data into train and validation (0.9-0.1 split).

## 4.1. Logistic Regression (Baseline)

The Logistic Regression Model was the worst performer amongst all of the models we tested. It was only able to get accuracy of 0.5 and ROC AUC score of 0.524.

## 4.2. Decision Tree

The Decision Tree classifier was able to perform better, the best model with num samples at leaf 40 gave ROC AUC score of 0.659 and accuracy of 0.61. For the detailed comparison of hyperparameters of the model, we refer to our notebook.

## 4.3. Random Forest Classifier

Our next model was the Random Forest classifier. For a detailed comparison of the hyperparameter tuning we refer to our notebook. Eventually, we get a respectable ROC AUC score of 0.706 and accuracy 0.647 on the test data for the model which used 50 estimators.

## 4.4. LGBM

Figure 2 indicates how the number of leaves (x-axis) of the tree in the model influences the train and test accuracy (y-axis). As the number of leaves increases train accuracy also shows an increasing trend whereas test accuracy increases up to a certain value and then it almost stays constant. Training accuracy increased from 0.64 to 0.72, whereas test accuracy increased from 0.64 till 0.66 for 100 leaves and remains constant with further increase in number of leaves. This effect of number of leaves on accuracy indicates that very high number of leaves would cause overfitting and low number would result in underfitting.

From the hyperparameterization of `num_leaves` and `min_data_leaf`, we found out that model with `num_leaves` 100 and `min_data_leaf` 100 gives the best performance with test and train accuracy of 0.6626 and 0.7082 (Figure 2) respectively. As per Figure 4 we see the detailed performance of LGBM (slightly higher NPV than PPV [13]). We also analyze feature importance in Figure 3. We can see that `SmartScreen`, `AVProductStatesIdentifier`, `AvSigVersion` (AntiVirus related) were the most important.

## 4.5. Neural Network

Finally we analyze the performance of our neural models. We developed 9 different models based on several hyperparameters which are described in Section 3. The best model (Model 6 in notebook) gave ROC AUC of 0.716 and accuracy of 0.65 on test data, which was significantly lower than the LGBM (0.728).

Note: For Extensive analysis of all models with hyperparameter tuning, we refer to our notebook.

# 5. Feature ablation study

After pre-processing there are 76 features in the data. Even though there are a high number of fea-

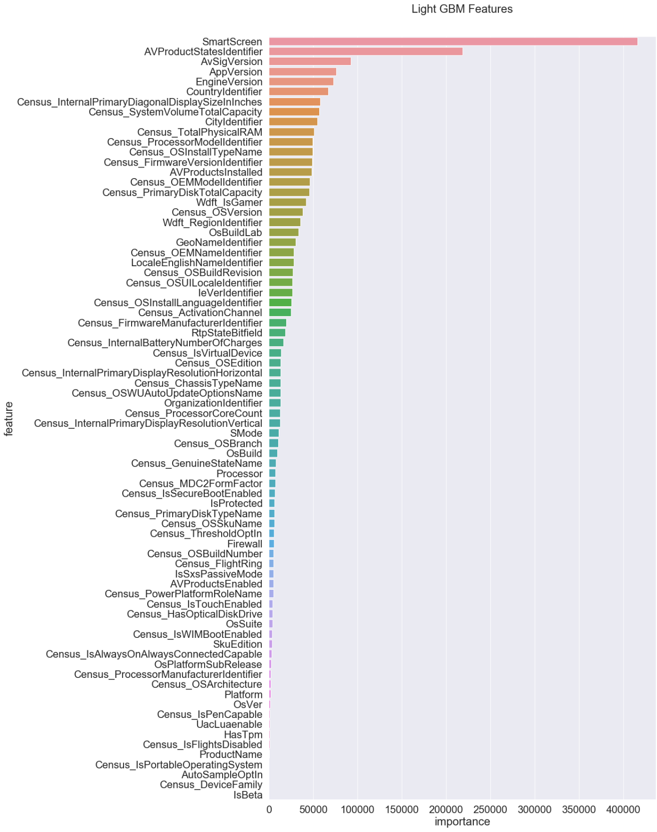

Figure 3: LGBM Features by Importance (Gain)

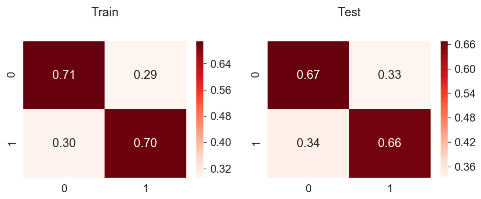

Figure 4: LGBM Output Confusion Matrix

tures, there was no dimensionality issue because the ratio of number of features to the total number of samples is very low. But to understand whether all features are really important, we dropped six more (`IsBeta`, `AutoSampleOptIn`, `Processor`, `Census_IsPortableOperatingSystem`, `OsVer`, `Census_OSSkuName`) features, based on the feature importance and correlation scores, and trained and evaluated the same models with reduced features data .

From Figure 7 we observe that reduced features set data does not affect the performance considerably. This shows that it is possible to build the model with less features without affecting model performance.

## 6. Discussion and Conclusion

So as part of this project, we tried to come up with suitable models for predicting the malware vulnerability of systems. We perform data analysis, feature selection and engineering, model comparisons and conclude that the LGBM is the best model suited for this task as per mentioned analysis. As seen in Figures 5, 6 it gave the higher accuracy, far superior ROC AUC score (0.728 compared to Neural Net 0.716) than the others and easily beats the baseline Logistic Regression model (AUC 0.524). Moreover we can get an interpretable output as we can visualize which features were more important.

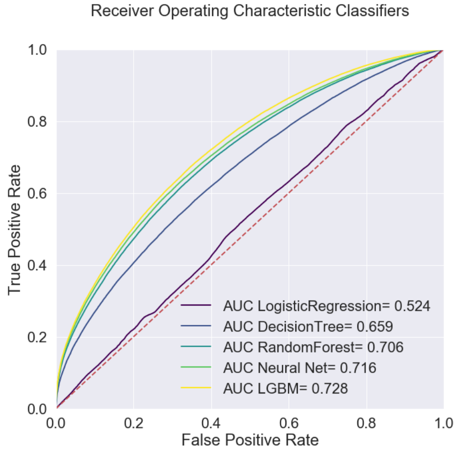

Figure 5: ROC AUC of Classifiers on Test Data

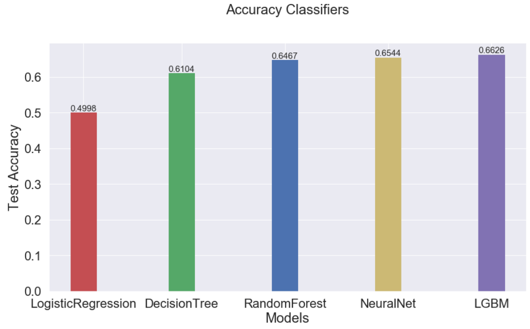

Figure 6: Accuracy of Classifiers on Test Data

Our results perhaps show why ensemble tree based methods have been quite popular for Kaggle competitions. Not only are they are fast to train, so easier hyperparameter tuning, but they can give competitive score even without extensive tuning. It also seems that such models are better suited for tabular heterogeneous data as ours.

## 7. Future Enhancements

Our results and feature importance showed that not all parameters are imperative. Thus, by performing smarter feature engineering we could improve the models further. Boosting based methods especially LGBM has extensive set of hyperparameters. Thus, we could further do intensive hyperparameter tuning for better performance. Finally, Malware analysis can also be treated as a time series analysis. Some of the provided features could give us some time information (perhaps the Engine versions) using which better models can be built.

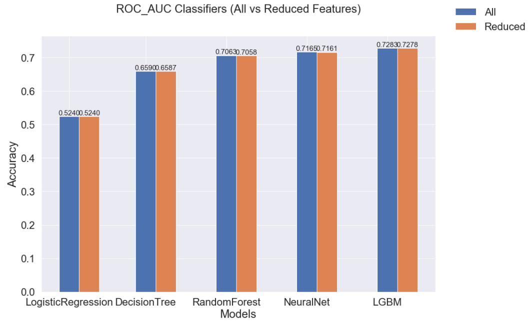

Figure 7: Reduced Features(Models vs ROC AUC)

## References

[1] J. Z. Bakdash, S. Hutchinson, E. G. Zaroukian, L. R. Marusich, S. Thirumuruganathan, C. Sample, B. Hoffman, and G. Das. Malware in the Future? Forecasting of Analyst Detection of Cyber Events. *arXiv e-prints*, page arXiv:1707.03243, July 2017.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[4] 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. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[6] P. Mandot. What is LightGBM, How to implement it? How to fine tune the parameters? Medium Blog Link.

[7] Microsoft. LGBM Hyperparameters. Hyperparameter Tuning Guide.

[8] Microsoft. LGBM Model. LGBM API Link.

[9] Microsoft. Microsoft malware prediction dataset. Kaggle Competition Link.

[10] N. M. Nawi, W. H. Atomi, and M. Rehman. The effect of data pre-processing on optimized training of artificial neural networks. *Procedia Technology*, 11:32 – 39, 2013. 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013.

[11] V. Prokopev. Mean (likelihood) encoding for categorical variables with high cardinality and feature interactions: a comprehensive study with python. Blog Link.

[12] M. Satnalika. On learning embeddings for categorical data using keras. Medium Blog Link.

[13] Wikipedia. Positive and negative predictive values. Article Link.

[14] Wikipedia. Random forest. Article Link.

[15] Wikipedia. Receiver operating characteristic. Article Link.