MALWARE DETECTION ON WINDOWS MACHINES USING MACHINE LEARNING APPROACHES

Ganesan Narayanan, Hongyu Cai, Kaiwen Tang, Swasthi Rao, Xinyu Qiu

Columbia University, COMS W4995 AML Group 22

ABSTRACT

In recent years, malware infections have posed significant challenges for Windows-based systems. The ability to predict the likelihood of a malware infection accurately can significantly aid in identifying current system vulnerabilities and averting potential damages associated with such infections. In this project, we explored various machine learning methodologies using the Microsoft Malware Prediction dataset to forecast the probability of infection. Among the models tested, the Neural Network demonstrated superior performance, achieving a test accuracy of 0.61.

1. INTRODUCTION

In the landscape of cybersecurity, Windows operating systems continue to be highly susceptible to malware attacks, which compromise system integrity and data security. As these threats evolve, there is a critical need for advanced predictive tools that can anticipate and mitigate potential breaches.

In this project, we would like to investigate the possibility of leveraging Machine Learning technologies to predict the possibility of a Windows machine being infected by various families of malware, based on different properties of that machine. We chose to use the Microsoft Malware Prediction dataset, which contains various properties for a collection of Windows machines and the ground truth of whether each machine is infected by malware. Using this dataset, we plan to train four machine learning models: XGBoost, Random Forest, K Nearest Neighbour (KNN), and Neuro Network, and investigated their performance on classifying whether a machine is infected by malware.

2. DATA ANALYSIS

2.1. Microsoft Malware Prediction Dataset

The Microsoft Malware Prediction dataset contains records for 8921483 Windows machines. For each machine, 82 features are collected from telemetry data such as Windows diagnostic reports. All the features are related to properties of a windows machine, such as the available RAM, OS version, type of malware detection application installed, the location of the machine, type of CPU used, etc. Every machine is uniquely identified by a *MachineIdnetifier*. The *HasDetection* column reveals the ground truth of whether a machine is infected.

This is a balanced dataset. Because its size is significantly large, we decided to use only 1% of the provided training data to train the machine learning models. These data are randomly sampled and has a total of 89215 entries.

2.2. Data Cleaning

Even though the dataset provides an extensive set of features to leverage, it requires substatial cleaning as lost of features contain missing values. This is mainly due to that the hardware and software configurations for Windows machines varies. To remove the unnecessary null entries, we performed the following:

1. Removed all features that has over 1000 null values. This resulted in the removal of 17 features, most of which are identifiers of software and/or hardware.

2. Removed all remaining rows that still contain Null values. This resulted the removal of 5201 rows.

The dataset after cleaning contains 84014 rows of data.

2.3. Preliminary Investigation

We categorized all features into 4 categories: identifiers: (unique identifiers of hardware and software components, such as the CPU), categorical variables, numerical variables, and target variable (the *HasDetection* column). We excluded all identifiers from the dataset because of their limited predictive power on the target variable. Because the large amount of features this dataset has, we also selected a set of highly important features based on our belief of their relations with the target variable and performed data analysis.

2.4. Data Visualization

We performed separate analysis on each type of features. For category features, we mainly analyzed their distribution across the target feature. A lot of the features related to version #s or OS builds were very skewed in distribution, with a couple of classes dominating the distribution. Other features such as binary variables indicating if features such as secure boot was enabled were more balanced in their distributions of class and against the target variable.

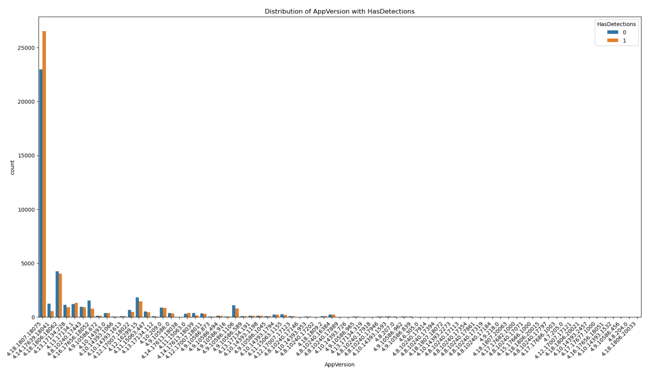

Fig. 1. Distribution visualization of one categorical variable

We also performed similar analysis on numerical features. We used a kernel density estimate to smooth the distribution. Our analysis revealed that a lot of the numerical features are clustered around a small range of values, but the overall domain is quite large as there are a few samples that extend much beyond the rest of the domain, which may mean we should remove outliers when training our model.

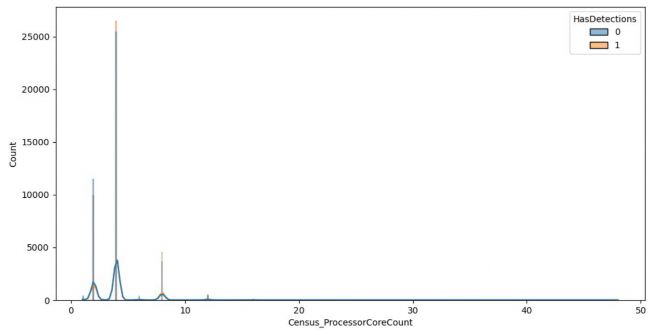

Fig. 2. Distribution visualization of one numerical variable

For the selected important features, we analyzed their correlation with the target variable. Unfortunately, the important features demonstrated low direct correlation with the target variable, indicating that these features may have complex mutual effects on the outcome.

3. MODEL IMPLEMENTATION

Based on our analysis, we performed One-Hot Encoding for the categorical features and Standardization for the numerical variables. Then, we separated our data into development set and test set using a 80/20 split. For each model, we first performed hyperparameter tuning with cross-validation

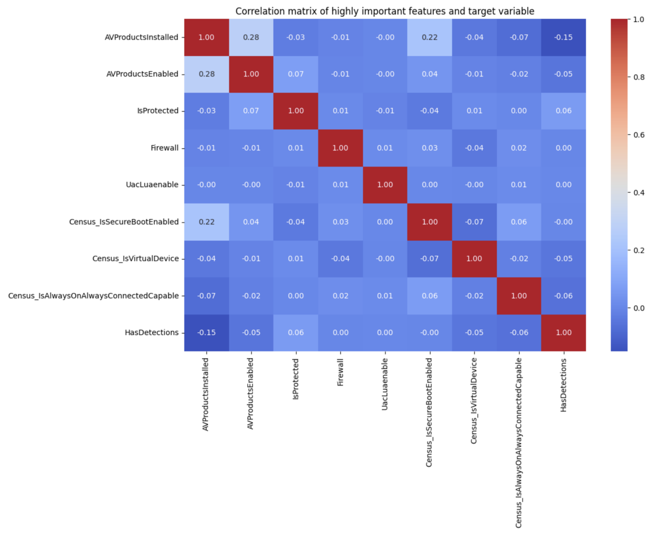

Fig. 3. Correlation matrix of highly important features and the target variable

on the development set and then tested the performance of the optimally-configured model on the test set. The optimal hyperparameter configuration for each model is:

- **Random Forest**: estimators = 200, max features = 8
- **XGBoost**: estimators = 100, max depth = 6, learning rate = 0.1
- **KNN**: number of neighbours = 28
- **Neural Network**: a hidden layer with 256 nodes $\rightarrow$ a dropout layer with 0.4 dropout possibility $\rightarrow$ a hidden layer with 128 nodes $\rightarrow$ a dropout layer with 0.2 dropout possibility $\rightarrow$ a output layer with one node. All hidden layers uses the ReLU function as activation function. The output layers use Sigmoid as activation function. The network is trained with the binary cross-entropy loss function and optimized by the Adam optimizer with a learning rate of 0.00023870368157235175.

4. RESULTS

The model training results are shown in Figure 4. We used the overall prediction accuracy and the precision, recall, and F-1 score for the positive (has malware) class to evaluate the performance of each model. To our surprise, all four models demonstrated similar test statistics. XGBoost and Neural Network performed the best with identical test statistics. Both of these models demonstrated a good performance in detecting all positive cases among the test data.

Model	Accuracy	Precision	Recall	F-1
Random Forest	0.58	0.58	0.57	0.58
XGBoost	0.61	0.60	0.67	0.63
KNN	0.58	0.57	0.61	0.59
Neural Network	0.61	0.60	0.67	0.63

Fig. 4. Model test statistics

5. DISCUSSION

5.1. Random Forest

The optimal random forest model achieved an accuracy of 58% and a weighted f1-score of 0.58. This performance is decent, given the inherent difficulty in the nature of the problem of preemptively predicting whether a computer would be susceptible to malware or not. The precision and recall scores were similar to each other at 0.58 as well, indicating that the model was balanced and not biased towards false positives nor false negatives. The most important features were also determined from the model, found to be the ones related to **processor core count**, **# of AV Products installed**, and **the type of disk installed**. The decent but not great performance may be due to the high dimensionality of the features. Although we removed some of the unnecessary features, with over 300 features still, the random forest model may be prone to overfitting, resulting in decreased performance.

5.2. XGBoost

The best XGBoost model achieved a test accuracy of 61.07%, with precision scores of 0.63 for non-malware instances and 0.60 for malware instances, and recall scores of 0.55 and 0.67, respectively. These results highlight its relative effectiveness in classifying both types of samples. The model's predictive capabilities were enhanced through hyperparameter tuning using RandomizedSearchCV, focusing on key parameters such as 'n_estimators', 'max_depth', and 'learning_rate'. Additionally, early stopping was implemented to prevent overfitting by halting training when no further improvement was observed.

XGBoost demonstrated an improved performance compared with Random Forest. It was also one of the best performing models among the four models investigated. However, we can observe that its classification capability is quite limited. Other than the reasons mentioned above, we also suspect that this sub-optimal performance is due to the fact that most of features used does not have a directly relationship with the presence of malware (as shown in the data analysis section, that features have low correlation with the target). Tree-based models may not have enough capacity to capture such vague relationships between features.

5.3. KNN

The analysis of the K-Nearest Neighbors (KNN) model for detecting malware on Windows machines shows a test accuracy of 57.97%, which is insufficient for high-stakes environments like malware detection. The confusion matrix shows significant false positives (3771) and false negatives (3292), indicating the model's poor ability to correctly identify malware. With precision values of 0.59 for non-malware and 0.57 for malware, and recall values of 0.55 and 0.61, respectively, the model's performance in distinguishing between malware and non-malware is mediocre. Factors contributing to this suboptimal performance include inadequate feature selection, the model's simplicity, imbalanced training data, and insufficient parameter tuning. Enhancements could include using more complex models, better feature engineering, ensuring data balance, and comprehensive parameter optimization to improve the accuracy and reliability of malware detection on Windows platforms.

5.4. Neural Network

The optimal neural network model achieved an accuracy of 61% and an F-1 score of 0.63, but struggled with overfitting as evidenced by increased loss in the validation set after only 5 or 6 epochs. Without dropout or batch normalization layers, accuracy was low and validation loss high. We tested both one-hidden-layer and two-hidden-layer configurations, which performed similarly. Opting for a structure with two hidden layers and a dropout layer following each, we aimed to better capture the non-linear relationship between outputs and variables. Hyperparameters were optimized using KerasTuner, similar to RandomizedSearchCV, focusing on the number of neurons, dropout probability, and learning rate. This model outperformed others, indicating superior learning of non-linearities, although the overall results were still not fully satisfactory, likely due to missing key features or insufficient information in the remaining features for effective classification.

6. CONCLUSION

In this project, we explored the feasibility of using machine learning algorithms to detect the presence of malware on Windows machines. We relied on the Microsoft Malware Prediction dataset and trained four models: Random Forest, XGBoost, KNN, and Neuro Network. The best prediction accuracy we obtained is 0.61, which can be considered as suboptimal. Future studies should be carried out to improve the model's performance. The main consideration should be improving the predictive capabilities of the features used, as the current features demonstrated insufficient capability in accurately depicting the presence of malware. We should also consider increasing the model's capacity to capture more nuanced relations between features.