Microsoft Malware Prediction

Team: For Want of a Better Name

Aniruddh K Budhgavi
iMT2018008
International Institute of Information Technology, Bangalore
aniruddh.kishore@iiitb.org

Omkar Lavangad
iMT2018053
International Institute of Information Technology, Bangalore
omkar.lavangad@iiitb.org

Tanmay Joshi
iMT2018527
International Institute of Information Technology, Bangalore
tanmay.joshi@iiitb.org

***Abstract* — Malware is a major threat for personal and enterprise computer users alike. Ransomware alone has now become a billion-dollar industry.[1][2] Over time, as a response to this, a robust antivirus industry has developed. NortonLifeLock (formerly Symantec), one of the largest antivirus corporations in the world, had revenues in excess of a billion USD in FY 2020[3]. However, malware continues to be a persistent and continuously evolving challenge, with new threats emerging everyday.**

Using readily available telemetry information and standard machine learning tools, we have generated a model using which one can predict if a machine is at risk of malware. Our findings are applicable to computers running Microsoft Windows.

***Index terms* — Feature Selection, One Hot Encoding, Label Encoding, K Fold, Cross Validation, Receiver Operating Characteristic, Confusion Matrix, Naive Bayes, Logistic Regression, Support Vector Classifier, Random Forest Classifier, Gradient Boosting, XGBoost, LightGBM**

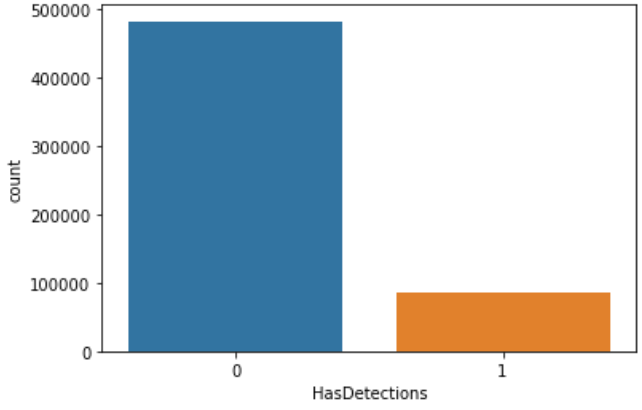
1. DATASET

The dataset is derived from the Microsoft Malware Prediction competition data available on Kaggle[4]. The data includes hardware and software configuration information for 550,000 computers, with each computer's information mapped to a unique identifier. There are 82 columns in the data, most of which are categorical in nature. The target variable is *HasDetections*.

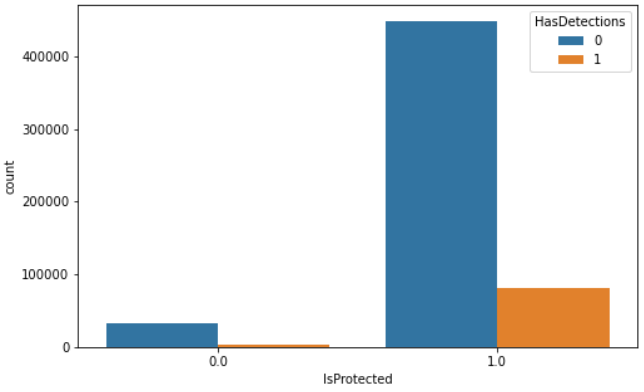
A full description of the feature columns is available at Kaggle.

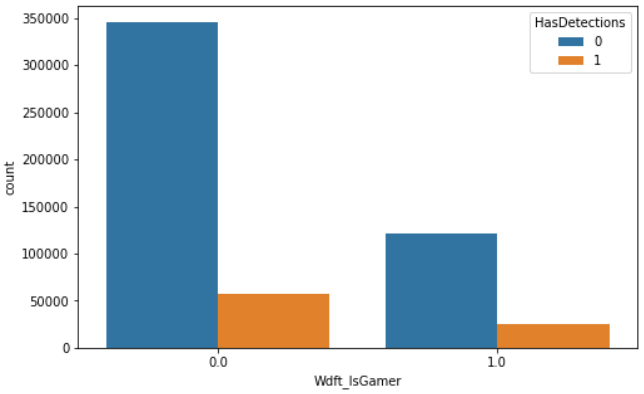
2. INITIAL OBSERVATIONS

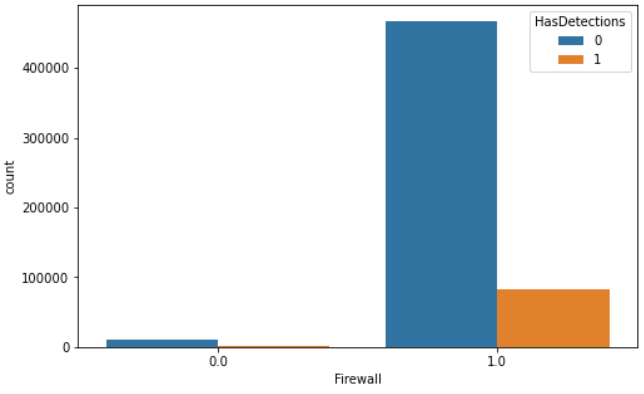
There is an imbalance in the target variable.

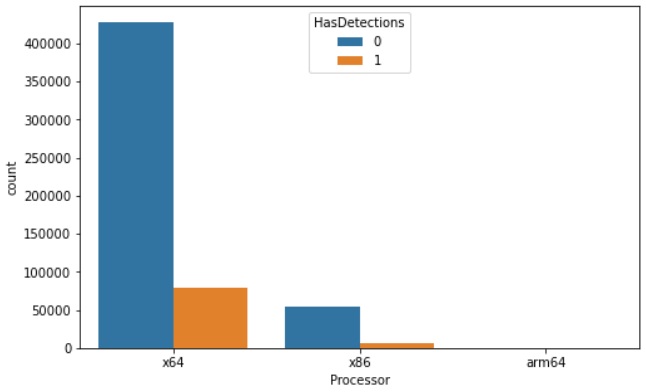


There is a similar imbalance in several of the features. Some representative plots are shown below.









These imbalances add new challenges to training.

3. DATA PREPROCESSING

3.1 NULL VALUE IMPUTATION

Most machine learning algorithms perform poorly or fail to run at all in the presence of missing data. Imputation of missing values is necessary. Given that many features are skewed towards one or the other category, we felt that it would be prudent to impute the missing values to the mode i.e. the majority class.

A few exceptions are those where the null values indicate not an absence of data, but perhaps where the feature is not applicable for the particular computer. For instance, the column *Census_InternalBatteryType* is irrelevant to desktop computers. Hence, it is safe to impute the null values in that column to a new category "None".

3.2 CATEGORICAL FEATURE ENCODING

Many of the columns in the dataset consist not of integers or floating-point numbers, but of alphanumeric character strings (henceforth referred to as string columns, for brevity). Given that most machine learning algorithms are applicable only to numeric data, it is necessary to perform a conversion from strings to numeric data.

With the exception of *MachineIdentifier*, the number of unique values in a string column is far lower than the total number of data points. Thus, the string columns are categorical in nature.

Two methods exist for encoding categorical features: Label Encoding and One-Hot Encoding. Both are available as tools in the Scikit-Learn library[5][6]. Here, we illustrate their working with an example.

Consider a column labeled *Fruit*. It has three unique values, *Apple*, *Mango*, and *Orange*.

Fruit
Apple
Mango
Orange
Orange
Mango
Apple

3.2.1 LABEL ENCODING

Each unique value in the column is mapped to an integer. In the above example, we could use the following scheme:

Old Value	New Value
Apple	0
Mango	1
Orange	2

Thus, the column is transformed as follows:

Fruit (old)	Fruit (new)
Apple	0
Mango	1
Orange	2
Orange	2
Mango	1
Apple	0

Label encoding is memory-efficient -- in general, the memory occupied by the data is unchanged or reduced before and after label encoding. However, label encoding is unsuitable for algorithms like Logistic Regression, Support Vector Machines and Naive Bayes. This is because an implicit ordering is created between the categories even if such ordering is not present in the data. However, we observed no major deterioration in performance in tree-based models and ensembles (Decision Trees, Random Forests, Gradient Boosting) when using label encoding as opposed to one-hot encoding.

3.2.2 ONE-HOT ENCODING

Here, for each unique value in the column, an indicator column is assigned. When a value occurs, a '1' is assigned to the corresponding column and a '0' is assigned to the remaining columns. For example:

Fruit (old)	Apple	Mango	Orange
Apple	1	0	0
Mango	0	1	0
Orange	0	0	1
Orange	0	0	1
Mango	0	1	0
Apple	1	0	0

In many applications, one-hot encoding is favourable because no implicit ordering of categories is assumed. However, one-hot encoding is very memory intensive *unless* the resulting table is stored as a Scipy Compressed Sparse Row Matrix (CSR matrix). CSR matrices are not supported by all model implementations. This is a trade-off which must be evaluated during model selection.

We tested both label encoding and one-hot encoding. There was a decrease of less than 0.01 in the scoring metric (described below) with label encoding when compared to one-hot encoding.

4. SCORING METRIC: ROC AUC

The Receiver Operating Characteristic curve is rumoured to have been developed during World War II for radar systems. It was utilized for analyzing images of chest X-rays[7].It has now been adapted for machine learning[8]. The ROC curve measures the performance of the model *at different threshold values for detection,* by plotting the true positive rate (TPR) against the false positive rate (FPR) at different probability thresholds. To understand TPR and FPR, we go back to the confusion matrix, which is:

		Prediction	
		0	1
Ground truth	0	True negative (TN)	False positive (FP)
	1	False negative (FN)	True positive (TP)

True Positive Rate (TPR) = True Positives/Ground truth positives
= TP/(TP + FN)

False Positive Rate (FPR) = False Positives/Ground truth negatives
= FP/(TN + FP)

The ROC curve for a model which makes random guesses (henceforth called a random model for brevity) is a straight line. This is easy to see. For a random model, irrespective of the threshold, the TPR will always be equal to the FPR -- this can be verified by writing the TPR and FPR in terms of probability, and applying the condition that the prediction is independent of the truth value.

For any model which is better than random guessing, the ROC curve will be above the line for a random model. I.e the true positive rate will, in general, be better than the false positive rate.

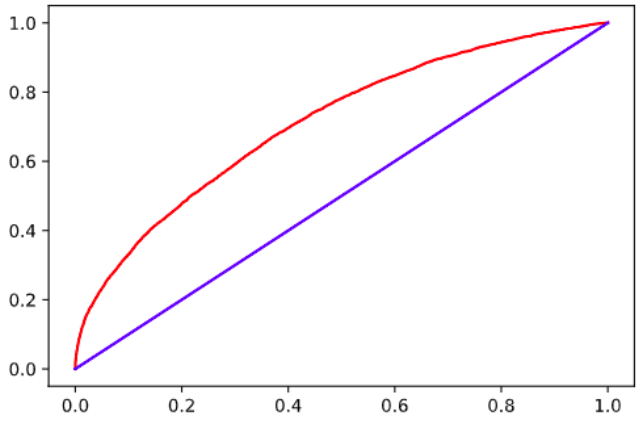


Figure 1: The ROC curve for our best model, on the cross-validation set. The X axis is the FPR while the Y axis is the TPR. The red curve is the ROC curve for our model, while the blue curve is the ROC curve for random guessing.

The area under the ROC curve (ROC AUC or AUROC)[9] is a popular metric for evaluating the performance of models. For models better than random guessing, the AUROC score will be higher than 0.5 -- the "perfect" model would have an AUROC score of 1.

5. MODEL SELECTION AND TRAINING

We tested a variety of different models under 10-fold cross validation[10] -- meaning that the model is trained on 90% of the available training set and validated against the remaining 10% -- for 10 different partitions. The most promising algorithm is Gradient Boosting as implemented in the LightGBM library[11]. We experimented with XGBoost[12] as well, but abandoned it shortly on account of excess training time. Apart from LGBMClassifier and XGBClassifier (which are from the LightGBM and XGBoost libraries respectively), all results are using the Scikit-Learn[13] implementations.

We experimented with our best models using oversampling methods from the Imbalanced-Learn library[14]. These experiments yielded no improvements in the scoring metric.

A summary of our results:

Model family/library	Mean cross validation AUROC score
LGBMClassifier	**0.71**
XGBClassifier	0.70
BernoulliNB	0.63
DecisionTreeClassifier	0.67
RandomForestClassifier	0.68
LogisticRegression	0.66
LinearSVC → CalibratedClassifierCV	0.64

6. CONCLUSION:

Our model is an effective method to identify machines at risk of malware. Such models may be useful for future identification of machines at risk of malware.

ACKNOWLEDGEMENTS

We would like to thank Professor G Srinivasa Raghavan, Professor Neelam Sinha as well as all the Teaching Assistants -- Amitesh Anand, Arjun Verma, Bukka Nikhil Sai, Divyanshu Khandelwal, Kotha Tejas, Mohd Zahid Faiz, Saurabh Jain, Shreyas Gupta, Tanmay Jain, Tushar Anil Masane, and Vibhav Agarwal.

We would like to thank the other teams in the contest for motivating us to pursue excellence. A special mention goes to team VSM (Shubhayu Das, Veerendra Devaraddi, Sai Manish) for productive ideation and inspiration.

REFERENCES

1. Ross Brewer, Ransomware attacks: detection, prevention and cure, Network Security, Volume 2016, Issue 9, https://doi.org/10.1016/S1353-4858(16)30086-1
2. Steve Mansfield-Devine, Ransomware: taking businesses hostage, Network Security, Volume 2016, Issue 10, https://doi.org/10.1016/S1353-4858(16)30096-4

3. NortonLifeLock Annual Report 2020, https://investor.nortonlifelock.com/About/Investors/financial-information/Annual-Reports/default.aspx
4. Kaggle Contest for AI511 ML course 2020, IIITB, https://www.kaggle.com/c/Malware-detection-NS/overview
5. Label Encoder from SKLearn documentation, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
6. One-Hot Encoder from SKLearn documentation, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html
7. Lusted, L. B. (1971). Signal Detectability and Medical Decision-Making. Science, 171(3977), 1217–1219. https://doi.org/10.1126/science.171.3977.1217
8. Andrew P. Bradley (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms, https://doi.org/10.1016/s0031-3203(96)
9. Scikit-Learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
10. Scikit-Learn documentation, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
11. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: a highly efficient gradient boosting decision tree. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3149–3157. https://papers.nips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf
12. Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794. DOI: https://doi.org/10.1145/2939672.2939785
13. Scikit-learn: Machine Learning in Python. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay; 12(85):2825−2830, 2011. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf
14. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Guillaume Lemaître, Fernando Nogueira, Christos K. Aridas; 18(17):1−5, 2017. https://jmlr.org/papers/volume18/16-365/16-365.pdf