Malware Detection

Team: SotA

Karthik Hegde
IMT2018509
IIITB
karthik.hegde@iiitb.org

Jishnu V K
IMT2018033
IIITB
vinodkumar.Jishnu@iiitb.org

Aman Kumar
IMT2018006
IIITB
aman.kumar@iiitb.org

I. Introduction

Cyberattacks and malware are one of the biggest threats on the internet.

Malware is shorthand for malicious software. It is software developed by cyber attackers with the intention of gaining access or causing damage to a computer or network, often while the victim remains oblivious to the fact there's been a compromise.

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.

Is there a way to protect more than one billion machines from damage by predicting whether the particular machine can be attacked by Malware?

Malware can wreak havoc on a computer and its network. Hackers use it to steal passwords, delete files and render computers inoperable. A malware infection can cause many problems that affect daily operation and the long-term security of your company.

Computer Economics recently conducted a survey of IT security professionals and managers on the frequency and economic impact of malware attacks on their organizations

Financial Impact of Malware Attacks 1997–2006

	Worldwide Impact (U.S. $)
2006	$13.3 Billion
2005	14.2 Billion
2004	17.5 Billion
2003	13.0 Billion
2002	11.1 Billion
2001	13.2 Billion
2000	17.1 Billion
1999	13.0 Billion
1998	6.1 Billion
1997	3.3 Billion

Source: Computer Economics, 2007 *Figure 1*

Fig. 1. Loss incurred due to Malware

We see that there is a huge worldwide loss due to Malware Attacks.So we need a way to protect machines from damage.

II. Dataset

The dataset used here is a part of Malware Detection Prediction Competetion which is hosted on Kaggle. The training data has 82 columns and the HasDetections column tells us whether the Windows machine's is infected by various families of malware, based on different properties of that machine. The training dataset is slightly imbalanced with 85% 0's(No malware detected) and 15% 1's(Malware detected). Also in the train dataset we have nearly 5.5lakh records which we will use to build ML Models

The testing data has 82 columns and we need to predict the HasDetections column based on the values of other 82 columns using Machine Learning algorithms.

Here 0 means not infected and 1 means infected.

Given the unique values in HasDetections column, it's understandable that the problem is a binary classification problem.

Here is the link to the dataset

https://www.kaggle.com/c/iiitb-ml-malware-detection-tanmay/data

III. EDA

Our EDA consists of three things - analysis of the target column, Examining missing values in the data, identifying the column types, and classifying them into numerical and categorical variables.

We first analyzed the target column - HasDetections, and as shown in the figures 2 and 3, it was heavily imbalanced.

The Missing value analysis indicated some seven feature columns:

- DefaultBrowsersIdentifier
- PuaMode
- Census_ProcessorClass
- Census_InternalBatteryType
- Census_IsFlightingInternal
- Census_ThresholdOptIn
- Census_IsWIMBootEnabled

whose more than 50% values were missing and we decided to drop these before moving ahead.

While analyzing the different columns, we came across several important observations. The first thing we noticed was several columns were imbalanced as well. Examination of categorical columns indicated few features - 'MachineIdentifier',

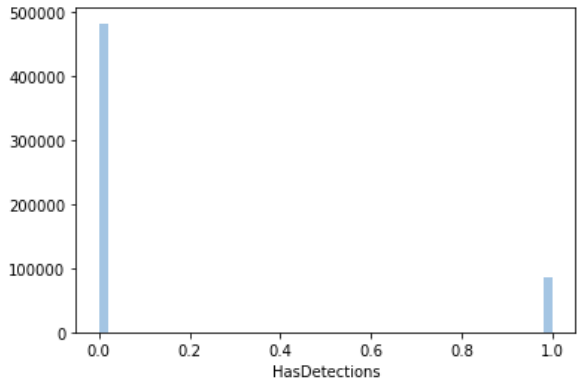

Fig. 2. Target column countplot

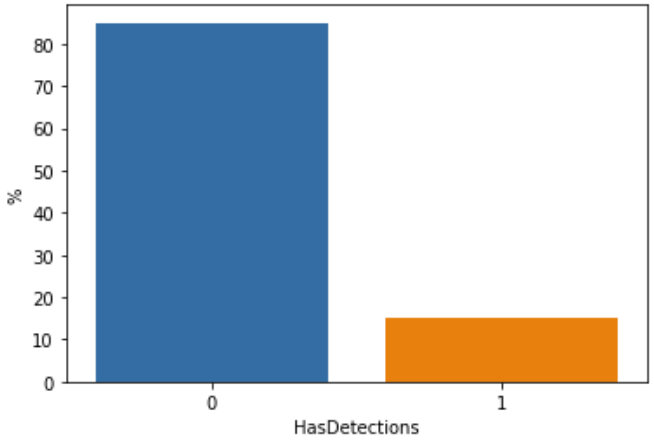

Fig. 3. Target column percentage wise plot

'AvSigVersion', 'OsBuildLab', 'Census_OSVersion' - that had more than 300 unique categorical classes (unique values). These columns were not considered for the training of the model. Examination of numerical columns showed 26 of them had less than 20 unique values. With this observation, these columns were considered to be categorical and were encoded before training.

IV. Pre-processing

In the stage of pre-processing, we dropped those columns that showed more than 50% of missing values. Then we moved to the stage of encoding the categorical columns. We decided to encode all of the categorical columns via the Label encoding scheme. While encoding, we encountered yet another observation that some of the columns had categories present in the test set but absent in the train set. Since the frequency of such unknown categories was too small (one or two), they were simply added to the label encoder list of categories. The next step we took was the imputation of the missing values in the remaining columns. The encoding of the categorical columns had already assigned a label to the missing values. The imputation of numerical columns was carried out by filling with the mean values.

V. Feature Engineering

The feature engineering scheme of Normalization was not applied as the model implemented was tree-based gradient boosting (XGBoost), which is robust to scaling. The only feature engineering step was to pass the top 12 important feature columns, extracted from the tuned first XGBoost model without smote, to a polynomial transformer of degree 2. These polynomial feature columns were then passed to the model along with the other remaining columns.

VI. Model Selection, Cross Validation and Training Process

We decided to try a few different models initially to see baseline performances initially before committing to the best one. The models we tried include logistic regression with smote, svm with smote, adaboost with and without smote and xgboost with and without smote. We divided the dataset into a train test split to evaluate all the models. Of all the models, we found that xgboost without smote was giving us the best score by a significant margin.

So we decided to try hyperparameter tuning on this model. Since the dataset is quite imbalanced and limited, we decided to use the k-fold cross validation technique to tune the model. By doing this, we saved some useful data for training at the expense of computational time. We manually tuned the hyperparameters after reading the documentation to see which hyperparameters would be useful for us. We feel that by manually tuning instead of using a grid search, we saved time and were able to make more dynamic adjustments depending on the cross validations scores.

We initially started with 100 estimators. We tuned the learning rate, tree depths, L2 regularization and positive weight scaling. By this method, we were able to acheive a public test score of 0.713.

After this, we decided to increase the number of estimators to 1000 to try and improve our score. These models took a lot more time to train but promised better results. By tuning, we were able to achieve a public test score of 0.717.

For our final model, we took the 12 most important features according to the classifier, and passed them through a polynomial transformer. We trained the model on these features along side the already existing features. Using this, we were able to get a public LB score of 0.71995 which translated to 0.72061 on the private LB.

VII. Conclusion

We were able to make a fairly accurate model to predict the risk of malware attack. The main challenge we faced were the fact that the dataset was quite skewed and the information regarding one class was limited. More data from this class would likely improve the performance of any model on this task. Using machine learning to predict malware attacks can prove quite helpful for anti-virus sofware development and the personal computer industry in general.

VIII. Acknowledgement

We would like to thank Professor Raghavan and Professor Neelam for introducing us the field of machine learning and a wide range of methods used. We would also like to thank the Teaching Assistant team for their valuable sessions and insights in practical machine learning, especially in dealing with imbalanced datasets.

References

[1] Documentation for XGBoost. Link: https://xgboost.readthedocs.io/en/latest/python/python_api.html

[2] Article on XGBoost tuning. Link: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[3] Documentation and user guide on sklearn cross validation. Link: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation