Malware Prediction

Shanthan Reddy. G
IMT2018522
Shanthan.Reddy@iiitb.org

Madhav. P
IMT2018524
sai.madhav@iiitb.org

Sriman Reddy. P
IMT2018525
sriman.reddy@iiitb.org

***Abstract*—Through this assignment, we are presenting a model that predicts if a machine (personal computer) is infected by malware based on different properties of that machine that were given to us. We have used several classification models such as LightBGM classifier and XGB classifier. The main goal of our assignment is to increase the accuracy for malware prediction.**

***Index Terms*—Feature Engineering, One-hot encoding, LGBM-Classifier,XGBoost,Ensembling**

I. Introduction

Malware is a malicious software designed to infiltrate and damage devices without the consent of the user. Once the system is infected, it can cause a lot of damage to the consumer. We might not have the capability to predict the probability of a malware infection before it even happens. So, it is important to identify the factors that increase the risk of being infected and take necessary precautions.

II. DATASET

For this assignment, we have used the Kaggle dataset on Malware Prediction. Each row corresponds to a machine identified by a unique MachineIdentifier. Using the information given in train.csv we have to predict the values of HasDetections in test.csv. There are a total of 567730 machines in our training set and a total of 243313 machines in the testing set. We are provided with 81 features(columns).

III. Observations and Preprocessing

A. Missing Data

- The first step in preprocessing is to check for null values. We have checked all the features for null values and found out that *PuaMode, Census_ProcessorClass, DefaultBrowsersIdentifier, Census_IsFlightingInternal, Census_InternalBatteryType, Census_ThresholdOptIn, Census_IsWIMBootEnabled* were having a very high percentage of NaN values.
- For features with data type as integer and object, the missing values were filled with the mode of those columns. For features with data type as float, the missing values were filled with the mean of the data.

Feature	Unique_values	null count	Percentage of missing values	type
PuaMode	1	567624	99.981329	object
Census_ProcessorClass	3	565528	99.612140	object
DefaultBrowsersIdentifier	582	538202	94.798936	float64
Census_IsFlightingInternal	2	469837	82.757120	float64
Census_InternalBatteryType	28	399998	70.455674	object
Census_ThresholdOptIn	2	358394	63.127543	float64
Census_IsWIMBootEnabled	1	357850	63.031723	float64
SmartScreen	13	207376	36.527222	object
OrganizationIdentifier	44	175893	30.981805	float64
SMode	2	38516	6.784211	float64

Fig. 1. Features with highest null count

B. Too-skewed features

We then checked for skewness in the features and found that *Census_OSBuildRevision, Census_InternalPrimaryDiagonalDisplaySizeInInches, Census_SystemVolumeTotalCapacity, Census_PrimaryDiskTotalCapacity,Census_TotalPhysicalRAM* features are highly skewed. We applied log() to remove that skewness.

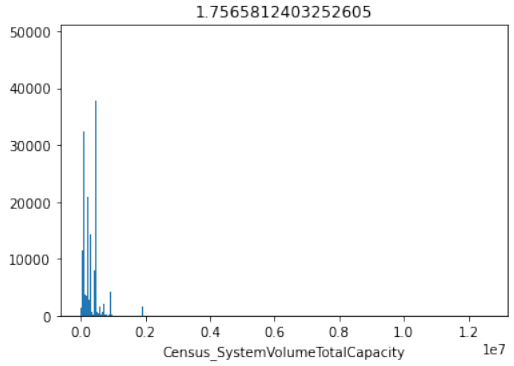

Fig. 2. Example of skewness in Census_SystemVolumeTotalCapacity feature

C. Removing unnecessary features

We then removed the features *PuaMode, Census_ProcessorClass, DefaultBrowsersIdentifier,Census_IsFlightingInternal,Census_InternalBatteryType,*

Census_ThresholdOptIn, Census_IsWIMBootEnabled, MachineIdentifier because more than 60% of the data in these features are missing. So we thought they were not very crucial and are unnecessary.

IV. Feature Encoding

Feature Encoding is the process in which we transform a categorical variable into a continuous variable for using them in our model. Since we are dealing with a lot of categorical data, this step is essential.

Ordinal encoding: In this encoding each unique categorical value is assigned to an integer. We use this technique when the categorical feature is ordinal i.e when the categories have an inherent order, which we don't have.

Target encoding: Target encoding correlates labels directly with the target.In target encoding for each category in the feature label is decided with the mean value of the target variable on a training data. Since some of the categories have a lot of unique count, overfitting occurs if we use this method.

Hence we have used **One hot encoding** to encode our categorical features. One hot encoding is a technique in which categorical features are converted into a form that could be provided to our Machine learning algorithms for better prediction. It essentially represents our categorical variables as binary vectors.

After encoding all the 26 categorical features, we created a sparse matrix out of the remaining int and float type features. Using hstack we horizontally stacked this sparse matrix with the previous sparse matrix that we got after one hot encoding.

V. Model Selection

LightGBM: LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages: Faster training speed and higher efficiency. Lower memory usage. Better accuracy.

XGBoost: XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

RandomForest: The random forest is a classification algorithm consisting of many decision trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. Since Random Forest generates a lot trees, it takes a lot of time to train.

We tried different ensemble classifiers like GradientBoostingClassifier, RandomForestClassifier and XGBClassifier in our models. Gradient boosting algorithms such as LightGBM and XGBoost, can process large datasets and less memory to run. Initially we built a LightGBM classifier which gave less accuracy. Furthermore, we examined the significant parameters that affect the model and figured out which ones give better results which includes the hyperparameter tuning that was considered. We observed better accuracy to a little extent with the tuning of hyperparameters after several trails.

In the process of tuning these hyperparameters we decided to try out random forest which can implement an algorithm without tuning the hyper-parameters and still can provide better results. We implemented RandomForestClassifier after the data preprocessing. We realized that this is utilizing a lot more memory compared to LGBM and even the accuracy wasn't much better compared to LGBM though there wasn't much difference in the runtime. So, due to the performance (in terms of accuracy) and memory issues, we then shifted back to the LGBM classifier and implemented XGBoost.
The following parameters were used for implementing LGBMClassifier:

- n_estimators=1000
- max_depth=10
- num_leaves=512
- colspample_bytree=0.3
- learning_rate=0.05

The following parameters were used for implementing XGBClassifier:

- n_estimators=1000
- max_depth=7
- num_leaves=75
- colspample_bytree=0.3
- learning_rate=0.12

ENSEMBLING: An ensemble method is a technique in which we use more than one model to make a better prediction or increase accuracy or even boost the performance. Every model's prediction is influenced by bias, variance and noise. In order to combat these disadvantages, we are using ensembling.

From *sklearn.metrics* we imported *roc_auc_score* to find the best possible weights for our models. Higher auc score implies better accuracy for the parameters. We ended up with a weight of **0.565** for LightGBM and a weight of **0.435** for XGBoost.

Ensemble Classifier	Accuracy
Light GBM	71.250%
Light GBM with Random Forest	70.981%
Light GBM with XGBOOST	71.466%

Fig. 3. Accuracies with different models

VI. Conclusion

We would like to conclude that we were able to come up with an efficient model to predict if a machine will soon be hit with a malware. Such projects have potential scope to protect more than one billion machines from damage before it happens.

CHALLENGES AND FUTURE SCOPE

After participating in this challenge, we got a better understanding of the commonly used approaches to solve the Malware Prediction challenge. As a future scope, we can compress these models and implement them on laptop and mobile applications which will save billions of devices from malware attacks. For this we need to have better accuracy. We want to try other possible ML algorithms to solve such problems. We will focus on making our model efficient for both time and space consumptions. We will also focus on tuning the hyperparameters to achieve better results.

References

[1] Microsoft Malware Prediction Competition", Microsoft Corporation, [online]. Available: https//www.kaggle.com/c/microsoft-malware-prediction

[2] "Features", LightGBM Documentation. [online] Available: https://lightgbm.readthedocs.io/en/latest/

[3] Gavrilut D., Cimpoesu M., Anton D., Ciortuz L., "Malware Prediction Using Machine Learning", International Multiconference on Computer Science and Information Technology, 2009 .

[4] A. Natekin and A. Knoll, "Gradient boosting machines," [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full/ [Accessed: November 2020].

[5] Available: https://en.wikipedia.org/wiki/Exploratory_data_analysis [Accessed: November 2020].

[6] S. Chatterjee, "Good Data and Machine Learning - Towards Data Science," [Online]. Available: https://towardsdatascience.com/data-correlation-can-make-or-break-your-machine-learning-project-82ee11039cc9/

[7] Random Forest, From Statistical Shape and Deformation Analysis, 2017 https://www.sciencedirect.com/topics/engineering/random-forest