Machine Learning Assignment

Team: Titans

Members: Mudit Goyal, Parithimalan A, Nachiappan S K

Malware Prediction

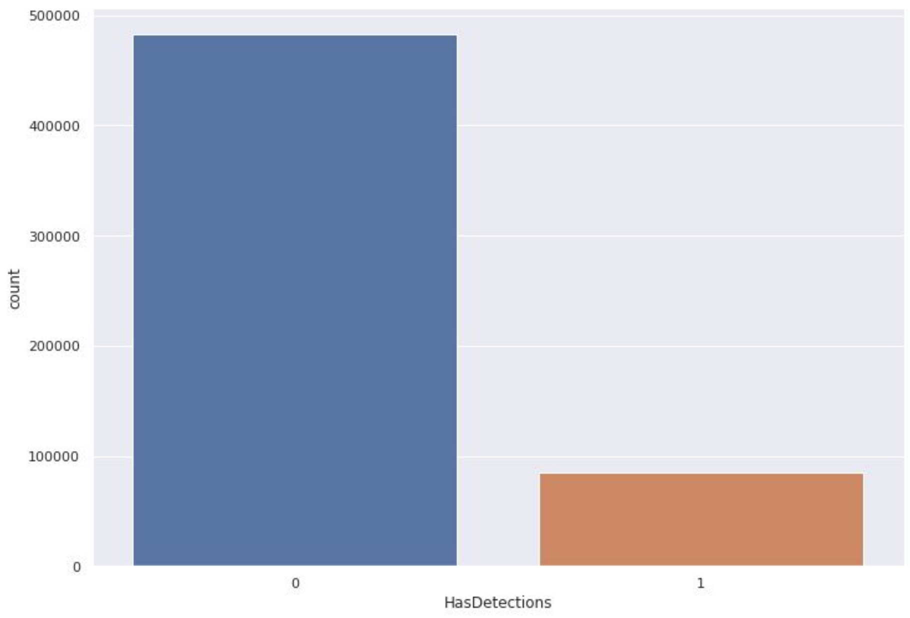
Problem Statement

Given various data of the machines, the goal is to predict the probability of whether a machine is infected by various families of malware.

Dataset Analysis:

The dataset consists of two sets: **train** and **test**. Both sets contain 80 features that contain information regarding the different machines and their properties. We have a separate column which is the primary key: MachineIdentifier which can uniquely identify a machine in the dataset. In the training dataset, we also have a separate column/dependent feature:

HasDetections which contains the ground-truth value whether the machine contains malware or not.



After analyzing the data we found that the data is highly imbalanced.

Data Pre-Processing 1:

This pre-processing is used for lightgbm.

- Data Imputation:

  On the basis of some plots and intuition, we have filled the nan values for both numerical and categorical data. Here are the following methods we have used to impute them:

  - Mean
  - Median
  - Mode
  - 0 (Zero)
  - Other/new category in case of the majority of the values are NaN

- Encoding:

  Since we have different unique categorical values in many categorical features in train and test sets. Instead of encoding them separately, we have merged the two sets and encoded them together to get much more accurate results.

  The train and test data frames are merged together with giving a value of 2 for the predictions. If a particular column in the data frame has less than 4 categories then it has been selected to be encoded using the one-hot encoding (maximum of n-1 i.e. 3 features has been created for each such feature)

  Since we don't want to increase the number of features by too much, we have just label encoded the other non-numerical fields.

- Redundant Features:

  We have removed:

  - Features having more than 70% of the nan values.
  - Features containing IDs or unique values for each machine
  - Features having the same values for all machines.

- Standardization/Normalization:

  Only features with more than 25 categorical values have been scaled down and features with range outside [-3,3] have been scaled.

Data Pre-Processing 2:

This pre-processing is used for xgboost and catboost. We have tried the previous type of preprocessing and tried different combinations of them but the best result we got was from a fairly simple preprocessing.

At first, when we tried to encode, train and test data there were inconsistent in the categories so we dropped such columns. Later we decided to merge the tables to encode and this method gave a significant increase to the score. To maintain consistency in data we merged the train and test table for the first three steps of the preprocessing and avoided it while scaling as scaling together will cause data leakage. This wouldn't happen while imputing values as they are completely different values and while label encoding it keeps consistency in data as there are some categories which are present in the test set which aren't present in the train set. So we have merged the tables for these steps.

- Data Imputation:

  We have replaced NaN values by assigning a new element in their place. For numeric data columns we have replaced nan with '-1' as no column had negative numbers   and for categorical data columns, we made a new category 'nan' and replaced it with NaN.

- Encoding:

  Since we have different unique categorical values in many categorical features in train and test sets. Instead of encoding them separately, we have merged the two sets and encoded them together to get much more accurate results.

  We have label encoded most all categorical features as they were mostly version and identity numbers so implicit ordering wouldn't affect them because they are in one way ordered. After trying both label and one-hot encoding on columns with fewer categories we found that label encoding gave the better result. So, we have label encoded all categorical features.

- Redundant Features:

  We found some features which have only 1 value in their column and removed them. After seeing the feature importance while running xgboost we have removed features that have feature importance less than 0.005.

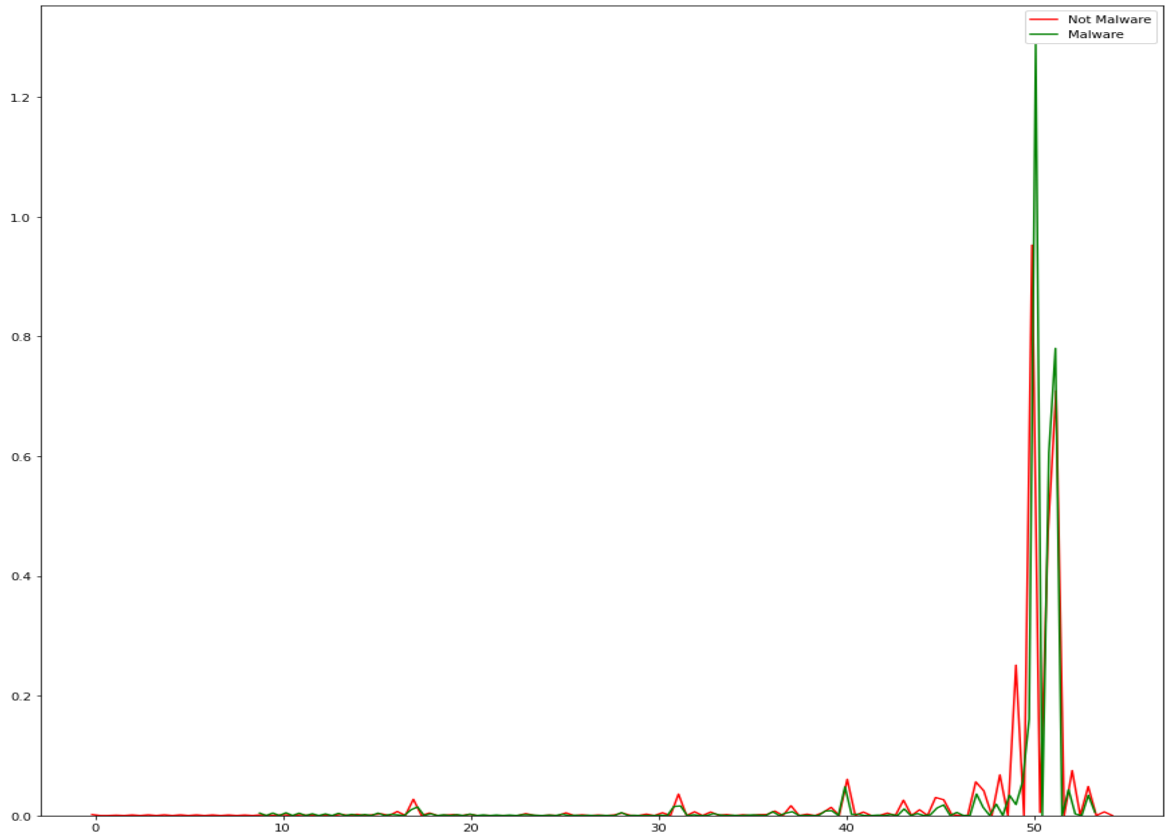
- Standardization/Normalization:

  Many features have values with varying ranges so we have scaled all the features using the Standard Scaler of sklearn library.

Feature Engineering:

- <u>New Feature:</u>

  By analyzing the feature importance we got from xgboost we created new polynomial features of degrees 2 and 3 on three features EngineVersion, AVProductsInstalled, SmartScreen but the only feature that increased the score was the square of EngineVersion. So we include one new feature that is square of EngineVersion.



As we can see there is a high probability of detecting malware in machines with EngineVersion < 50. We can generally say, that older versions are more probable to get infected.

- <u>Oversampling using SMOTE:</u>

  SMOTE stands for Synthetic Minority Oversampling Technique and is an oversampling technique used to increase the samples in a minority class. We have used this to increase the data samples of the minority class. This method gave good results in combination with xgboost and did not work well with other models.

  After trying different ratios for SMOTE we found that the ratio of minority class to the majority class of 0.21 gave the best results. The original ratio was 0.1765.

- <u>Undersampling:</u>

  We did not use undersampling as we might lose some valuable data points and it would also decrease the size of the dataset. Also, after using it, we did not get a better score.

Models:

After data preprocessing and feature engineering we have used the data on many models and the only models to give a roc_auc score of 0.7 and above were Catboost, XGboost, Lightgbm. We have used various metrics to check the performance of models. After picking these 3 models we have tuned the models by using various combinations of hyper-parameters to find the most suitable one for each model. We have tackled the imbalanced data using appropriate hyperparameters like scale_pos_weight which gave astonishing results.

Below we have mentioned the final set of hyperparameters used:

1. <u>Catboost:</u>

   HyperParameters:

learning_rate	0.1
l2_leaf_reg	250
border_count	100
scale_pos_weight	4

2. Xgboost:

   HyperParameters:

learning_rate	0.1
n_estimators	1000
scale_pos_weight	2.1

3. Lightgbm:

   HyperParameters:

learning_rate	0.01
num_leaves	255
max_bin	511
num_iterations	1000

Metrics:

Since we have an imbalance dataset roc_auc score.

We have analyzed the performance of the model for different hyperparameters using:

- Stratified Cross-validation with cv=5 and scoring parameter: roc_auc**.**

Ensemble:

We have done weighted averaging of the probabilities predicted by the models.

After trying different combinations we got the best result using

Y_final = Y_catboost*0.1 + Y_xgboost*0.4 + Y_lightgbm*0.5

Y_catboost,Y_xgboost,Y_lightgbm are the probabilities found by their respective models.
Y_final is the final probability used for submission.

Final Result:

MODEL	Public Score	Private Score
Catboost	0.71758	0.71967
XGBoost	0.71744	0.71912
LightGBM	0.71692	0.71761
Ensemble	**0.72122**	**0.72286**

References:

https://catboost.ai/docs/concepts/python-reference_parameters-list.html#python-reference_parameters-list

https://scikit-learn.org/stable/

https://lightgbm.readthedocs.io/en/latest/

https://xgboost.readthedocs.io/en/latest/

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

https://github.com/frenzytejask98/ML_TA_IIITB_2020