Computer Security Symposium 2020
26 - 29 October 2020

Gradient Boosting Decision Tree Ensemble Learning for Malware Binary Classification

Yun Gao[1,a) Hirokazu Hasegawa[2,b)] Yukiko Yamaguchi[3,c)]
Hajime Shimada[3,d)]

Abstract:
The increasing number of malicious software spread through the Internet has become a serious threat. Malware authors use obfuscation and deformation techniques to generate new types of malware in order to evade the detection of traditional detection methods, so that it is widely expected for machine learning methods that classifies malware and cleanware based on the characteristics of the samples. The current research trend is to use machine learning technology, especially decision tree technology, to identify new malicious software quickly and accurately. The purpose of this paper is to investigate malware classification accuracy based on latest decision tree based algorithms including ensemble learning. Therefore, we use the FFRI Dataset 2019 to construct malware detection models from surface analysis logs and PE header dumps. We have successfully developed a malware detection model that is more accurate than previous studies. We have obtained impressive classification results using only 27 features.

Keywords: malware classification,machine learning, ensemble learning

1. Introduction

With the feverish development of machine learning and artificial intelligence, there are many areas where AI has brought about great advances, such as image recognition and text sentiment analysis. Many applications have also arisen in the field of security that use AI for security protection and attacks.

Malware is software that causes damage to a single computer, server, or computer network. Incidents caused by one malware and its variants can cause millions of dollars in damage, i.e. WannaCry that exploits remotely exploitable EternalBlue attack has caused worldwide disasters, such as the shutdown of the Japanese Honda Motor Company [1]. Furthermore, malware is becoming more sophisticated and diverse every day to avoid malware detection schemes. Therefore, malware detection scheme is an important issue in cybersecurity, especially as more and more people in society become dependent on computing systems. Malware detection methods can be divided into static malware detection and dynamic malware detection [2]. Static methods classify samples as malicious or benign without executing samples, while dynamic methods detect malicious software according to its runtime behavior. In theory, dynamic malware detection allows for direct observation of malware actions that are not easily obfuscated and make it more difficult to reuse existing malware [3]. In fact, it is difficult to collect datasets of malware behavior because malware can identify sandbox environments and prevent itself from executing malicious actions. Moreover, dynamic malware detection requires many sandboxes to treat number of doubtful samples so that it is dramatically increasing detection cost. In contrast, while static malware detection is known to be undecidable in general [4], it is possible to create huge datasets by aggregating binary files, and identify malware before it executes.

In this study, we explored static malware detection accuracy with the latest gradient boosting decision tree frameworks and PE file features given by surface analysis result. Our results show that CatBoost achieved impressive evaluation results with only a few features and a short training time.

2. Related Work

Over the past few years, malware detection has evolved due to the gradual rise in the threat posed by malware to large enterprises and government agencies. Since 1995, various machine learning-based methods for static portable executable (PE) malware detection have been proposed [5–9]. Schultz et al. represented PE files by including features such as import functions, strings, and byte se-

1 Graduate School of Informatics, Nagoya University
2 Information Strategy Office, Nagoya University
3 Information Technology Center, Nagoya University
a) gaoyun@net.itc.nagoya-u.ac.jp
b) hasegawa@icts.nagoya-u.ac.jp
c) yamaguchi@itc.nagoya-u.ac.jp
d) shimada@itc.nagoya-u.ac.jp

quences [6]. Kolter et al. used techniques for byte-level N-grams and natural language processing, including TF-IDF weighting of strings, to detect and classify malicious files [7]. Saxe used histograms through the use of byte entropy values as input features and multilayer neural networks for classification [8]. Raff et al. showed that fully connected and recursive networks can be applied to malware detection problems [10]. They also use raw bytes of the PE file, and build an end-to-end deep learning networks [9]. Okamoto used XGboost for malware detection in SCIS 2019 [11]. They converted the categorical features to numerical features, but used a relatively large number of features in model training.

3. Machine Learning Frameworks based on Decision Tree

3.1 Decision Tree

In machine learning, a decision tree is a predictive model. It represents a mapping between an object's attributes and its values. Each node in the tree represents an object, each forked path represents a possible attribute value, and each leaf node corresponds to the value of the object represented by the path from the root node to the leaf node.

3.2 Ensemble Learning

Ensemble Learning is a method of combining several different base models into a single ensemble model. It reduces both the bias and variance [12] of the final model, thereby improving the score and reducing the risk of overfitting.

3.2.1 Bagging

The idea behind Bagging is that all base models are treated consistently, with only one vote in each base model's hand. A democratic vote is then used to get the final result. In most cases, bagging results in smaller variance. Bagging is a hugely popular ensembling method which is used in algorithms like Random Forest. It gains accuracy by not only averaging the models but also trying to create models that are as uncorrelated as possible by giving them different training sets [13].

Random Forest are a more advanced algorithm based on decision trees. Like a decision tree, a random forest can be used for both regression and classification. A forest is constructed in a random way and this forest is made up of many unrelated decision trees that are not related to each other.

3.2.2 Boosting

The fundamental difference between boosting and bagging is that the base model is not uniformly treated, but is constantly tested and filtered to select the "elite", and then the elite are given more votes, while the poor base models are given less votes, and then the final results are obtained by combining all the votes.In most cases, the boosting results are less biased.

Boosting is a sequential process, where each subsequent model tries to correct the errors of the previous model. Therefore the succeeding models are dependent on the previous models and we need to train the models in sequence instead of parallel.

Adaboost Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers. This is done by building models from training data and then creating a second model to try to correct errors from the first model. Add models until the training set is perfectly predicted or add the maximum number of models. AdaBoost is the first truly successful enhancement algorithm developed for binary classification.

Gradient Boosting is a method for implementing Boosting, the main idea of which is that each time a model is built, it is in the direction of a gradient decrease in the loss function of the previously built model.

Gradient Boosting Decision Tree is also known as MART (Multiple Additive Regression Tree). GBDT is an iterative decision tree algorithm that consists of multiple decision trees, and the conclusions of all the trees are added together to make the final answer.

XGboost is a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More importantly, it provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system [14].

LightGBM proposes two novel techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). With GOSS, it excludes a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain. With EFB, it bundles mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features. LightGBM speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy [15].

CatBoost is a GBDT framework with fewer parameters, support for category-type variables, and high accuracy based on symmetric decision trees as a base learner implementation, which addresses the pain point of processing category-type features efficiently and rationally. In addition, CatBoost solves the Gradient Bias and Prediction shift problems to reduce overfitting, thereby improving the accuracy and generalization of the algorithm [16].

3.2.3 Stacking

Stacking is an integrated learning technique that integrates multiple classification models or regression models through a single meta-classifier or meta-regressor. The base model uses the entire training set for training, and the meta-model trains the features of the base model as features.

Stacking was introduced by Wolpert in 1992 [17]. It is a method that uses k-fold for training base models which then make predictions on the left out fold. These so-called out-of-fold predictions are then used to train another model (the meta model) which can use the information produced by the base models to make final predictions.

All the weak learners in stacking are called level 0 learners, their output is accepted by a level 1 learner, and the final result is output. This is actually a hierarchical structure.

3.2.4 Blending

Blending is a word introduced by the Netflix competition winners [18]. It is very similar to stacking with the only difference being that instead of creating out-of-fold predictions using k-fold to create a small holdout dataset which will then be used to train the meta-model. In addition, the meta-model results are blended in different ways. Blending is linear blend, while Stacking is nonlinear blend.

4. Proposed Feature Selection

4.1 Dataset

We use the FFRI Dataset 2019 provided by the MWS Research Dataset [*1] [19]. This dataset contains 250,000 malware samples and 250,000 cleanware samples. In total, it contains 500,000 pieces of data obtained from the surface analysis. Each sample is a JSON file containing 7 fields: **file_size**, **date**, **hashes**, **lief**, **peid**, **trid**, and **strings** fields. Each field contains multiple layers of JSON data.

4.2 Features in Dataset

In maximum, we can use following number of features in FFRI Dataset 2019.

– From **file_size**: We can use 1 numeric feature.
– From **hashes**: We can use 13 numeric features.
– From **lief**: We can use 29 numeric features and 3 categorical features.
– From **peid**: We can use 10 categorical features.
– From **strings**: We can use 100 categorical features.

4.2.1 Numerical Feature

Numerical feature, which can be either continuous or discrete, is generally expressed as a real value. In general, decision tree type algorithms do not require preprocessing of numeric features.

As an example, in the lief.option_header.dll_characteristics field, 50.8% of the values are 0, 13.6% are 34112, and 19.3% are 320.

4.2.2 Categorical Feature

Categorical feature indicate a data point belongs to a certain class or has certain characteristics.

CatBoost automatically handles category features in a special way. First, it does some statistics on the category features, calculates the frequency of a category feature, and then adds hyper-parameters to generate new numerical features. With catboost, you don't have to handle category features manually anymore.

As an example, in the peid.Anti-Debug field, 47% of the values are "no", 35% are "yes", and 19% are "no (yes)".

4.3 Feature Selection

The FFRI Dataset 2019 provides 9 different fields of information,from which we selected 5 data fields. We used the information from fields **file_size**, **lief**, **peid**, and **strings** as the features of our dataset and the **label** field as the label of the dataset. The file_size field contains only one numeric data. The lief field is a multilayer structured JSON file parsed from the PE file using the LIEF library[*2]. This field extracts a total of 14 features, including 11 numeric features and 3 categorical features. The peid field is the information of the PE file parsed by PEiD[*3], it is a two-layer structured JSON file. We extracted 5 categorical features from peid field. The strings field is the string information contained in the PE file. We extracted the first 10 strings in the dataset, and the numbers that follow represent the order of the strings. Out of these 10 strings, we used 7 features of high importance around the beginning part because strings extracted around beginning part is not too varied, but vary moderately enough to usable as features. In the end, we selected a total of 27 features for the training data. The information used in this study is shown in **Table 1**.

Table 1 Features chosen as best.

features_name	features_types
file_size	numeric
lief.header.characteristics	numeric
lief.header.pointerto_symbol_table	numeric
lief.header.time_date_stamp	numeric
lief.header.numberof_sections	numeric
lief.optional_header.imagebase	numeric
lief.optional_header.checksum	numeric
lief.optional_header.sizeof_initialized_data	numeric
lief.optional_header.minor_linker_version	numeric
lief.optional_header.dll_characteristics	numeric
lief.entrypoint	numeric
lief.virtual_size	numeric
lief.sections	categorical
lief.data_directories	categorical
lief.optional_header.subsystem	categorical
peid.PEiD	categorical
peid.DLL	categorical
peid.Packed	categorical
peid.mutex	categorical
peid.Anti-Debug	categorical
strings_9	categorical
strings_7	categorical
strings_5	categorical
strings_8	categorical
strings_4	categorical
strings_10	categorical
strings_6	categorical

We also tried to select 157 and 69 features from the dataset as the model training set, respectively. Through experiments, we found that the gap between the evaluation indicators and the 27 features is not very large, indicating that a large number of them are not very important. For example, the hash information in the hashes field is explicitly not effective as a classification feature because it is varied between variable malware binaries. Some semantic hashes are also not particularly effective as classification features. So in the end, we only selected 27 features that are significantly useful for improving classification accu-

[*1] https://www.iwsec.org/mws/2019/about.html

[*2] https://github.com/lief-project/LIEF

[*3] https://github.com/K-atc/PEiD

Table 2 Quantity table of different feature combinations.

	156 features	69 features	27 features
file_size	1	1	1
date	0	0	0
hashes	13	0	0
lief	32	14	14
peid	10	7	5
trid	0	0	0
strings	100	47	7

racy. With the dramatic reduction of features, our training time for classifier generation was also reduced substantially. The quantity table of different feature combinations is shown in **Table 2**. The 27 features reduced the model TPR by approximately 0.002% and increased the FPR by 0.005% compared to the 156 features. However, the computation time 392s decreased to 136s so that we decisded 27 features as a final choise. 67 features gives intermediate results so that we did not choose it because the difference is so small.

4.4 Analysis Result of Feature Importance

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable.

To get the importance of the feature, CatBoost simply takes the difference in the measure (loss function) between the model used under normal circumstances (when we include the feature) and the model without the feature (the model was built using the original model that removed the feature from all the trees in the collection). The larger the difference, the more important the feature. Figure of CatBoost and LightGBM feature importance ranking is shown in **Fig. 1**. X-axis denotes features with score order and Y-axis denotes score (1.0 in maximum). As shown from figure, CatBoost and LightGBM gives difference importance order to features. Thus, we thought that we can compensate weak points of both method by stacking two methods.

5. Experiment

This section details the specifics of our experiment and results. Our experiments were conducted on Manjaro 20.0.0 Lysia system (Linux kernel version is 5.6.16-1) with a AMD Ryzen 7 3700X CPU, 32GB RAM, and a NVIDIA GeForce RTX 2070 SUPER GPU. In addition, the open source libraries used in the experiments are Python 3.7.6, Anaconda 4.8.4, CatBoost 0.23, LightGBM 2.2.3, and Scikit-learn 0.22.2. The code written for this evaluation will be opened on GitHub later.[*4]

5.1 Preprocessing

As a preprocessing, firstly, read the fields we need from the FFRI Dateset 2019 JSON lines file and parse the multi-layered JSON file structure layer by layer. Secondly, the parsed results are stored in a CSV file. We extracted a total of 27 features and the labels of each sample for our dataset. Finally, we used 5-fold cross-validation to split the dataset proportionally and equally into five copies, four of which were used as training sets and the remaining one as a test set. Since our experiments use the LightGBM and CatBoost frameworks, there is no need for complex feature engineering; numerical features can be used directly, and categorical features can also be used directly but need to be declared.

Table 3 LightGBM Baseline Parameters.

parameter_name	parameter_value
random_seed	2020
boosting_type	gbdt
objective	binary
metric	binary_logloss
n_estimators	38
learning_rate	0.1
num_leaves	123
colsample_bytree	0.8
subsample	0.9
max_depth	15
reg_alpha	0.1
reg_lambda	0.1
min_split_gain	0.01
min_child_weight	2

Table 4 CatBoost Baseline Parameters.

parameter_name	parameter_value
random_seed	2020
loss_function	Logloss
eval_metric	AUC
iterations	3000
od_type	Iter
depth	10
early_stopping_rounds	500

5.2 Models and Parameters

We use three open source libraries Scikit-learn[*5], LightGBM[*6], and CatBoost [*7]. Use GridSearchCV for autotuning in both CatBoost and LightGBM, respectively, to get the most optimized parameters.

LightGBM was open-sourced by Microsoft in 2017. A fast, distributed, high performance gradient boosting framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. The parameters of LightGBM in this study is shown in **Table 3**.

CatBoost is a depth-wise gradient boosting library developed by Yandex. It uses oblivious decision trees to grow a balanced tree. The same features are used to make left and right splits for each level of the tree. The parameters of CatBoost in this study is shown in **Table 4**.

Ensemble (LightGBM+CatBoost) We use Scikit-learn's StackingClassifier to combine the CatBoost and LightGBM models together. Category probability values generated by the first layer of LightGBM and the CatBoost base classifier as inputs to the meta-classfier and use logistic regression for the final category predictions. We use

[*4] https://github.com/kou18n/malware-detection-ensemble

[*5] https://scikit-learn.org

[*6] https://github.com/microsoft/LightGBM

[*7] https://catboost.ai

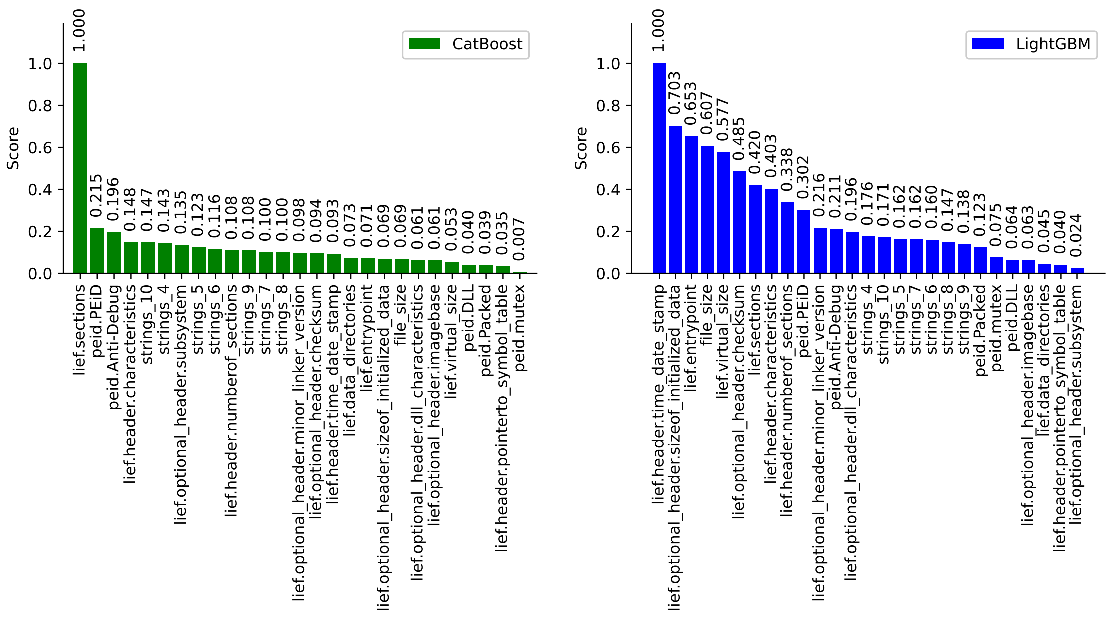

Fig. 1 Features Importance Ranking

the category probability values generated by the first layer of LightGBM and the CatBoost base classifier as inputs to the meta-classfier and use logistic regression for the final category predictions.

5.3 Evaluation Metric

True Positive Rate(TPR) Indicates the proportion of all positive samples that are currently allocated to the true positive sample.

False Positive Rate(FPR) Indicates the proportion of true negative samples that are currently misclassified into the positive sample category out of the total number of negative samples.

Area Under Curve(AUC) AUC is the area under the ROC curve, calculated as the calculus value of the ROC Curve, the meaning is: given a random positive and negative two samples, the probability of ranking the positive sample before the negative sample, so the larger the AUC, indicating that the positive sample is more likely to be ranked before the negative sample, that is, the better the classification amount results.

5.4 Experimental Results

The experiment was stratified 5-fold cross-validation with no shuffling and a fixed random seed of 2020. We counted the mean TPR, FPR, and AUC values for 5-fold cross-validation. The results of the three models after 1000 iterations are shown in **Table 5**. We can see that the combined result of our stacking model is significantly better than the other two models. The difference between the three models for AUC values is not very large, and the

Table 5 Experimental Results.

model_name	TPR	FPR	AUC
CatBoost	99.8680% (-0.0276%)	0.1104% (+19.48%)	99.99935%
LightGBM	99.6452% (-0.2506%)	0.2748% (+197.40%)	99.99432%
Stacking	99.8956%	0.0924%	99.99929%

model with the best TPR and FPR values is our stacking model. The TPR values for our stacking model were 0.0276% higher than CatBoost and 0.2506% higher than LightGBM. The FPR values were 19.48% and 197.40% lower than CatBoost and LightGBM, respectively.

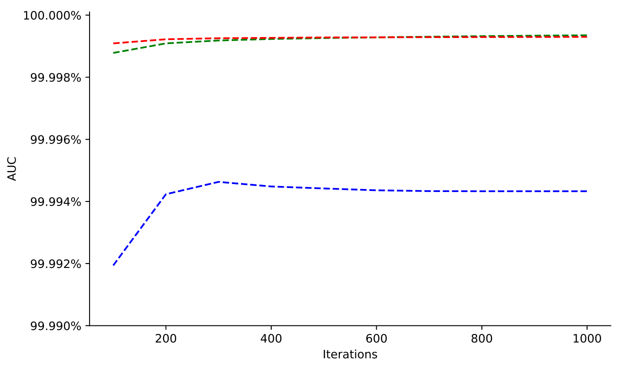

Fig. 2 Figure of AUC with iterations.

The AUC with iterations of our stacking model in comparison with the CatBoost and the LightGBM baseline models in shown in **Fig. 2**. X-axis denotes number of iterations and Y-axis denotes AUC. We examined iterations until 3000 but there is almost no difference after 1000 iterations so that the graph is terminated with 1000 iterations.

We found that the TPR and FPR values using our stacking model were significantly better than CatBoost and LightGBM regardless of the number of iterations performed. As the iteration increased, the results stabilized at 1000 iterations, where the TPR was 99.8956% and the FPR was 0.0924%. Relationship figure between TPR, FPR and iterations of three models is shown in **Fig. 3**. X-axis denotes number of iterations and Y-axis denotes both TPR and FPR. Note that the threshold value that separate malware/cleanware decision is differing from Fig. 2 so that ensemble shows explicit improvement from CatBoost.

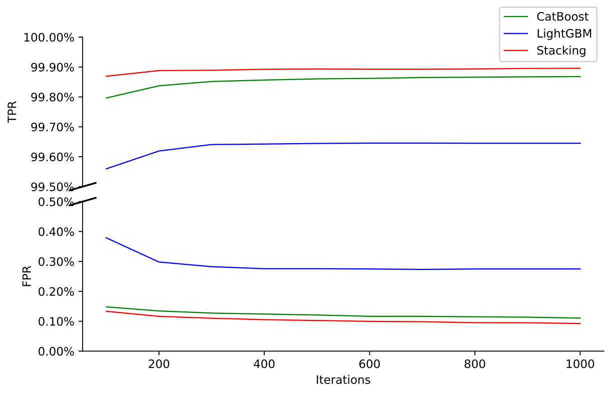

Fig. 3 Figure of TPR and FPR with iterations.

The Receiver Operating Characteristic curves (ROCs) of our stacking model and CatBoost models almost overlap, while the ROC of the LightGBM is at the bottom. The mean ROCs of our stacking model in comparison with the baseline models in shown in **Fig. 4**. X-axis denotes FPR with log scale and Y-axis denotes TPR.

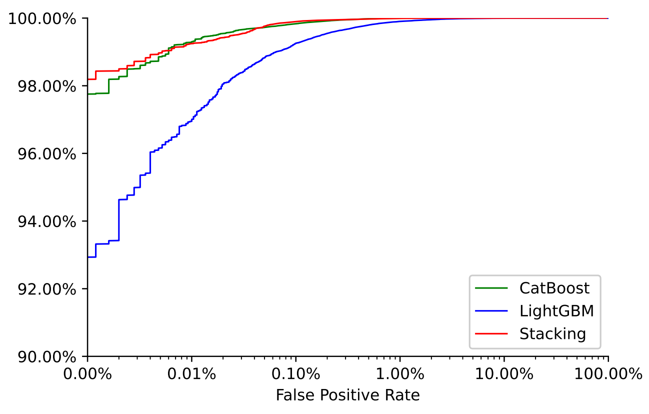

Fig. 4 Mean ROC curves with logarithmic X-axis scale.

We used 5-fold cross-validation, and the ROC curves for each fold are shown in **Fig. 6**. X-axis and Y-axis is identical to Fig. 4. The ROC curves for our stacking model and CatBoost has a relationship of win or lose in each validation. On the other hand, the ROC curves for LightGBM are consistently on the lower side.

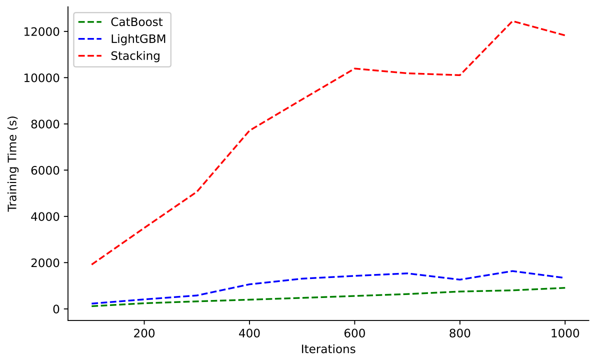

Fig. 5 Figure of training time with iterations.

The training of our stacking model is quite time-consuming, because it is a logistic regression model, which needs either be trained on the predicted class labels or probabilities from the ensemble. The difference between the Catboost model and the LightGBM model is not that big. CatBoost is trained with GPU, LightGBM cannot be accelerated with GPU, because the Bin size of LightGBM is too big due to the long categorical feature. Stacking also only can use CPU. The training time with iterations of our stacking model in comparison with baseline models in shown in **Fig. 5**. X-axis denotes iterations and Y-axis denotes training time with second. The training time is the aggregate time of the 5-fold croos validation.

6. Conclusions

In this study, we show the GDBT algorithm for malware classification. The performance of the LightGBM algorithm is relatively poor, and CastBoost has the best overall performance, but does not reduce the FPR value further.

We also apply a static PE malware detection method using the ensemble two GBDT algorithm. On the FFRI Dataset 2019, our proposed ensemble model has a better FPR than CatBoost and uses fewer feature vector dimensions than in previous studies, which reduces the training time and improves the model performance. In addition, the validation results also show that the GDBT algorithm can be used to solve the malware classification problem.

7. Future Work

We have used machine learning GDBT method for malware classification study, and FPR value is crucial for malware detection model, so we plan to reduce FPR value further. A follow-up study is planned to use deep learning methods for malware classification.

Acknowledgments We appreciate to all the members of the MWS community, especially for the dataset providers. This work was supported by JSPS KAKENHI Grant Number JP19H04108.

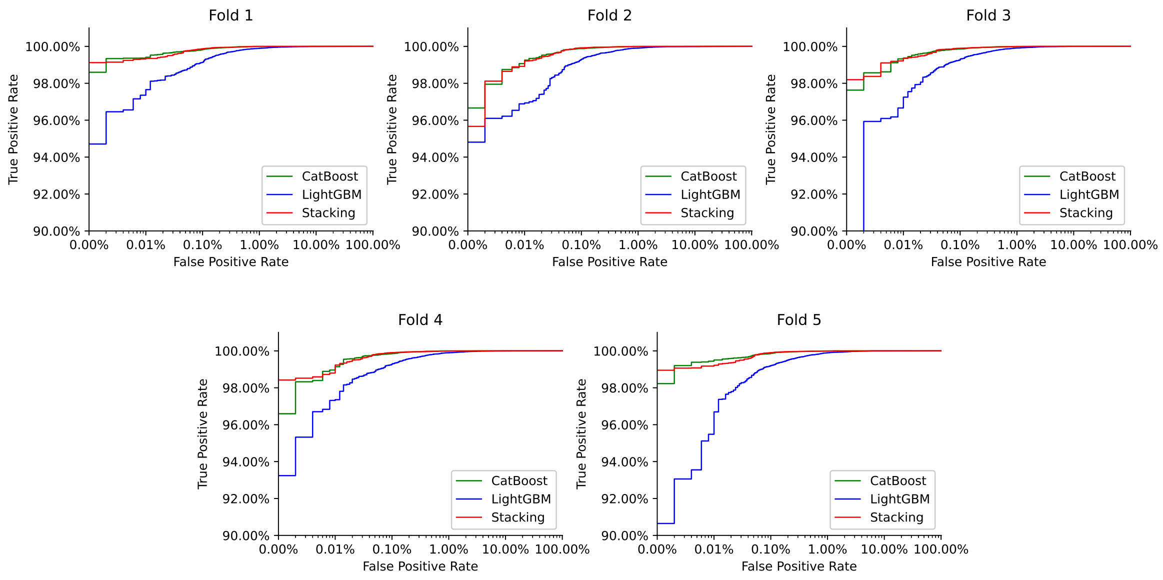

Fig. 6 ROC curves for 5-fold cross-validation.

References

[1] Qian Chen and Robert A. Bridges. Automated Behavioral Analysis of Malware: A Case Study of WannaCry Ransomware. In *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017*, pages 454–460. IEEE, 2017.

[2] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, 44(2):6:1–6:42, 2012.

[3] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference, ACSAC 2007,Miami Beach, Florida, USA, December 10-14, 2007*, pages 421–430. IEEE Computer Society, 2007.

[4] Fred Cohen. Computer viruses: Theory and experiments. *Comput. Secur.*, 6(1):22–35, 1987.

[5] Jeffrey O. Kephart, Gregory B. Sorkin, William C. Arnold, David M. Chess, Gerald Tesauro, and Steve R. White. Biologically Inspired Defenses Against Computer Viruses. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 985–996. Morgan Kaufmann, 1995.

[6] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. In *Proceedings of the 22nd IEEE Symposium on Security and Privacy, S&P 2001, Oakland, California, USA May 14-16, 2001*, pages 38–49. IEEE Computer Society, 2001.

[7] Jeremy Z. Kolter and Marcus A. Maloof. Learning to Detect and Classify Malicious Executables in the Wild. *J. Mach. Learn. Res.*, 7:2721–2744, 2006.

[8] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Proceedings of the 10th International Conference on Malicious and Unwanted Software, MALWARE 2015, Fajardo, PR, USA, October 20-22, 2015*, pages 11–20. IEEE Computer Society, 2015.

[9] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware Detection by Eating a Whole EXE. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, volume WS-18 of *AAAI Workshops*, pages 268–276. AAAI Press, 2018.

[10] Edward Raff, Jared Sylvester, and Charles Nicholas. Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, pages 121–132. ACM, 2017.

[11] Yohei Okamoto, Fumihiro Shindo, Wataru Matsuda, Mariko Fujimoto, Takuho Mitsunaga, and Hiroshi Masamoto. Malware detection using tree-based machine learning algorithms. In *Symposium on Cryptography and Information Security, SCIS 2019, Otsu-city, Japan, January 22-25*, 2019.

[12] Arjun Chandra, Huanhuan Chen, and Xin Yao. Trade-Off Between Diversity and Accuracy in Ensemble Generation. In Yaochu Jin, editor, *Multi-Objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*, pages 429–464. Springer, 2006.

[13] Leo Breiman. Bagging Predictors. *Mach. Learn.*, 24(2):123–140, 1996.

[14] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD 2016, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016.

[15] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st Conference on Neural Information Processing Systems, NIPS 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3146–3154, 2017.

[16] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. CatBoost: gradient boosting with categorical features support. *CoRR*, abs/1810.11363, 2018.

[17] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[18] Andreas Töscher and Michael Jahrer. The bigchaos solution to the netflix prize 2008. *Netflix Prize, Report*, 2008.

[19] Masato Terada, Mitsuaki Akiyama, Takahiro Matsuki, Mitsuhiro Hatada, and Yoichi Shinoda. MWS Datasets for Anti-Malware Research -Contribution to the community and its challenges- (in Japanese). *IPSJ SIG Technical Report*, 2020-IFAT-139(8):1–6, 2020.