An Empirical Evaluation of Automated Machine Learning Techniques for Malware Detection

Partha Pratim Kundu
I[2]R, A*STAR
Singapore
kundupp@i2r.a-star.edu.sg

Lux Anatharaman
I[2]R, A*STAR
Singapore
lux@i2r.a-star.edu.sg

Tram Truong-Huu
I[2]R, A*STAR
Singapore
truonght@i2r.a-star.edu.sg

ABSTRACT

Nowadays, it is increasingly difficult even for a machine learning expert to incorporate all of the recent best practices into their modeling due to the fast development of state-of-the-art machine learning techniques. For the applications that handle big data sets, the complexity of the problem of choosing the best performing model with the best hyper-parameter setting becomes harder. In this work, we present an empirical evaluation of automated machine learning (AutoML) frameworks or techniques that aim to optimize hyper-parameters for machine learning models to achieve the best achievable performance. We apply AutoML techniques to the malware detection problem, which requires achieving the true positive rate as high as possible while reducing the false positive rate as low as possible. We adopt two AutoML frameworks, namely AutoGluon-Tabular and Microsoft Neural Network Intelligence (NNI) to optimize hyper-parameters of a Light Gradient Boosted Machine (LightGBM) model for classifying malware samples. We carry out extensive experiments on two data sets. The first data set is a publicly available data set (EMBER data set), that has been used as a benchmarking data set for many malware detection works. The second data set is a private data set we have acquired from a security company that provides recently-collected malware samples. We provide empirical analysis and performance comparison of the two AutoML frameworks. The experimental results show that AutoML frameworks could identify the set of hyper-parameters that significantly outperform the performance of the model with the known best performing hyper-parameter setting and improve the performance of a LightGBM classifier with respect to the true positive rate from 86.8% to 90% at 0.1% of false positive rate on EMBER data set and from 80.8% to 87.4% on the private data set.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; **Intrusion/anomaly detection and malware mitigation**.

KEYWORDS

Malware detection, automated machine learning, hyper-parameter optimization

ACM Reference Format:

Partha Pratim Kundu, Lux Anatharaman, and Tram Truong-Huu. 2021. An Empirical Evaluation of Automated Machine Learning Techniques for Malware Detection. In *Proceedings of the 2021 ACM International Workshop on Security and Privacy Analytics (IWSPA'21), April 28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3445970.3451155

1 INTRODUCTION

Malicious software (malware) detection is an important problem in this current era of the growing sophistication of our digital world. Malicious individuals and/or organizations, using some clever means, put some malicious program(s) in the targeted computer systems to prevent the computer systems from performing their usual operations and/or to steal sensitive data. Malware detection is therefore a critical task to prevent any financial loss as well as any permanent damage to the organization's reputation. The emergence of machine learning (ML) and deep learning (DL) and their success stories in other disciplines such as computer vision, natural language processing, and robotics have motivated both industry and academia to adopt ML/DL to the malware detection problem [10, 13, 14]. Malware samples are analyzed by static or dynamic analysis techniques to extract useful features that are used to train ML models to learn the difference between malware and goodware (i.e., benign programs)[1]. The number of extracted features of each sample is large and goes up to a few thousand. The malware samples evolve very quickly[2]. To incorporate all variants of malware including latest ones, the data sets used for training ML models are significantly large, requiring not only long training time but also a large amount of computational resources before the models converge and can be used in the malware detection phase.

While there exists a great variety of complex ML models and each model has lots of hyper-parameters to tune to achieve its best performance, it is increasingly difficult even for an ML expert to follow all of the recent best practices into their modeling and their task is becoming even more difficult due to the fast development of state-of-the-art ML techniques. Thus, the AutoML frameworks offer an attractive and alternative paradigm for the novice who may not be aware of the intricacies of ML algorithms. Given sufficient computational and storage resources, AutoML frameworks allow ML practitioners to determine not only the ML model that works the best for a specific data set but also the best hyper-parameter

[1] Static analysis does not require executing malware samples but uses a reverse engineering tool to extract features from the files such as header information and byte sequences. In contrast, dynamic analysis needs to execute samples in a secure virtual box and captures their runtime behavior such as system calls.

[2] Every day, the AV-TEST Institute registers over 350, 000 new malware and potentially unwanted applications (PUA).

setting for the chosen model to yield the best performance. With AutoML frameworks, the best ML practices need to be implemented only once and then being repeatedly deployed to various problems. This allows ML practitioners to scale their knowledge to many problems without the need for frequent manual intervention for model selection, the selection of proper ensemble of models, hyper-parameter tuning, feature engineering, data pre-processing, etc.

In this paper, we adopt AutoML to the malware detection problem. We present an empirical evaluation of two AutoML techniques including AutoGluon-Tabular [6] and Microsoft NNI [12]. We use two malware data sets in our work to train a Light Gradient Boosted Machine (LightGBM) model. The first data set named EMBER [1] is a publicly available data set which is a labeled benchmarking data set for training ML models for malware detection based on static analysis of Windows portable executable (PE) files. With 2 million samples, this data set is fairly big, open, and general enough to cover several interesting use cases. The second data set named SecureAge contains recently-collected benign and malware PE files that we acquired from a local company[3]. In order to validate the malware detection model on the SecureAge data set, we first extract static features using the LIEF tool[4], which provides 2381 features for each sample. Training the detection models on these data sets require a significantly long time, up to 10 hours for a hyper-parameter setting. Thus, instead of manually performing hyper-parameter tuning, we demonstrate that the use of AutoML techniques could mitigate the complexity of the hyper-parameter optimization problem. Comparing the performance of the malware detection models obtained with the hyper-parameter setting yielded by AutoML frameworks with that of the known best performing or default setting, we see that a significant improvement has been achieved.

The rest of this paper is organized as follows. Section 2 discusses the state of the art of malware detection and AutoML. Section 3 describes the various AutoML frameworks and their operational schematics. We present the experimental results and discussion in Section 4 before concluding the paper in Section 5.

2 RELATED WORKS

2.1 Malware Detection

Various ML/DL techniques have been developed for malware detection in order to filter out the samples that show malicious behavior. In [7], Gavrilut *et al.* proposed a versatile framework, which employs different ML algorithms to successfully distinguish between malware samples and benign samples while aiming to minimize the number of false positives. The authors used a simple multi-stage combination (cascade) of different versions of the perceptron algorithm to achieve the goal. In [11], the authors proposed a machine learning-based malware analysis system comprising three modules namely data processing, decision making, and new malware detection. The data processing module deals with gray-scale images, opcode *n*-gram, and import functions, which are employed to extract the features of the malware. The decision-making module uses the features to classify the malware and to identify suspicious samples. Finally, the detection module uses the shared nearest neighbor (SNN) clustering algorithm to discover new malware families. In [5], David *et al.* presented a novel deep learning-based method named for automatic malware signature generation and classification. They used a deep belief network (DBN), implemented with a deep stack of denoising autoencoders, generating an invariant compact representation of malware behavior. The authors showed that signatures generated by the DBN were helpful for the accurate classification of new malware variants.

All of the work mentioned above require significant involvement of ML expertise to decide a proper set of hyper-parameters for the ML algorithms used. However, a hyper-parameter setting may work best only for a specific data set (i.e., data distribution). Given the fast evolution of malware samples, the detection models have to be frequently retrained with a tuned hyper-parameter setting to achieve the desired performance. It is worth mentioning again that retraining malware detection models and hyper-parameter tuning are a laborious and challenging task. Thus, we believe AutoML techniques will significantly contribute to the success of ML/DL not only in malware detection but also in other disciplines.

2.2 Proliferation of AutoML

While AutoML has not been adopted for malware detection (and cybersecurity in general), it has been widely used in healthcare and the Internet of Things (IoT). A huge amount of healthcare data is generated every day, thus requiring an automated process to generate a set of actionable knowledge from the data. ML models, especially DL models, are very good at doing so by improving patient safety, quality of care, and reducing healthcare costs. The usage of AutoML would allow the healthcare industry to build, validate, and deploy ML solutions rapidly, and therefore pass the benefits of improving the quality of healthcare to patients effectively [15]. AutoML techniques are also used to gain insights into the colossal amount of 60 ZB data generated due to the streaming of data between various IoT devices. To process this huge amount of data, human ML expertise is a must but mostly non-expert ML practitioners actually handle the whole operational ecosystem. Thus, AutoML techniques could be used to support these practitioners. One such AutoML framework is Decanter AI [4], which uses semi-supervised ML algorithms to precisely select the optimal algorithms to build a predictive model. It automates the data analytics workflow from feature engineering, algorithms selection, model building and parameters tuning by evaluating the problem at hand. When IoT devices stream data into this algorithm, it would efficiently refine the model for the user's data set. As IoT data that does not have labels, it automatically assigns classes for data with similar behaviors using clustering.

In this work, we demonstrate how AutoML is specifically useful for the malware detection problem. As discussed earlier, the field of malware detection is very dynamic due to the very frequent addition of novel malwares. To the best of our knowledge, we are the first to use AutoML in malware detection.

3 OVERVIEW OF AUTOML FRAMEWORKS

3.1 AutoGluon-Tabular

AutoGluon-Tabular is an easy and efficient Python framework for tabular data. It is developed by Amazon Web Services (AWS) and

[3] SecureAge: https://www.secureage.com/sg/
[4] Library to Instrument Executable Formats (LIEF): https://lief.quarkslab.com/

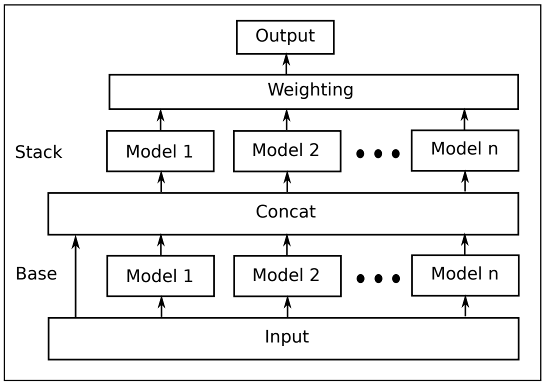

Figure 1: Operational schematic of AutoGluon-Tabular.

was open-sourced for public use in 2020. It allows us to train state-of-the-art ML models for well-known applications such as object detection, text and image classification, and tabular data prediction. Unlike existing AutoML frameworks that primarily focus on model and hyper-parameter selection, it succeeds by assembling multiple models and stacking them in multiple layers as shown in Figure 1. It can be run in Windows OS and Linux OS, using a workstation or from a terminal. It works effectively when running on Amazon elastic compute cloud (EC2) instances, as it can produce results faster by taking advantage of the quasi-unlimited computational resources of AWS. It provides a user-friendly API as it is designed in such a way that users can input the raw data set without any pre-processing and train ML models with a few lines of codes. It also allows the users to set time constraints to build a model and assures fault tolerance in case of any interruption.

AutoGluon supports algorithms such as Random Forests, Extremely Randomized Trees, K-nearest Neighbors, LightGBM, CatBoost, and a deep neural network (Multi-Layer Perceptron) model. AutoGluon shares similar design choices as the models presented in [3] and [9]. It applies a separate embedding layer to each categorical feature, where the embedding dimension is selected proportionally to the number of unique levels observed for this feature [8]. The individual embedding layers enable the network to separately learn about each categorical feature before its representation is blended with other variables for multivariate data. The embedding of categorical features is concatenated with the numerical features into a large vector, which is then fed into a 3-layer feed-forward network as well as directly connected to the output predictions via a linear skip-connection. It is the first AutoML framework to use per-variable embedding that is directly connected to the output via a linear shortcut path that can improve their resulting quality via gradient flow. Most existing AutoML frameworks instead just apply standard feed-forward architectures to one-hot encoded data.

3.2 Microsoft Neural Network Intelligence

Microsoft NNI is a toolkit to assist users to design and tune ML models, neural network architectures, or the parameters of complex systems, in an efficient and self-directing way. The properties like ease-of-use, scalability, and flexibility make it appealing to non-expertise ML practitioners. Using Python `pip`, a user could install it on her machine. It could be used as a command-line tool or as a WebUI. It has the capability to run parallel trails depending on the capacity of training platforms, to access remote machines and various training platforms (e.g., OpenPAI, Kubernetes). It allows users to customize various hyper-parameter tuning algorithms, neural architecture search algorithms, early stopping algorithms, etc. It also leverages early feedback to speed up the tuning procedure.

As shown in Figure 2, NNI contains several components. `Trial` is an individual attempt to run an ML model by applying a new configuration (i.e., a set of hyper-parameter values). `Tuner` is an AutoML algorithm that generates a new configuration to run as a new `Trial`. `Assessor` analyzes `Trial`'s intermediate results (e.g., periodically-evaluated accuracy on the test data set) to decide whether the trial is needed to be stopped early. An experiment starts running on NNI platform when `Tuner` receives a search space. It then generates a set of configurations that will be submitted to training platforms such as a local machine, a remote machine, or a training cluster. The performance is reported back to `Tuner` that in turn generates a set of new configurations to be explored in the next trial.

3.3 Operational Schematics of AutoGluon-Tabular and Microsoft NNI

Users can easily use AutoGluon for tabular data sets by setting a few operational parameters including:

- `time_budget`: the duration of AutoGluon will run to search for optimal parameters. The longer the time used for running, the larger the search space will be covered.
- `nthreads_per_trial`: the number of threads used to run for a trial, the higher the number of threads, the faster the algorithm will converge.
- `classifier`: the classifier used for malware detection.
- `evaluation metric`: the objective of the optimization problem, e.g., maximizing the true positive rate given a fixed false positive rate.
- `search strategy`: AutoGluon supports different search strategies including random search and Bayesian optimization search.

Users do not need to specifically define a search space for AutoGluon. This is a great advantage for non-expertise ML practitioners. Similar to AutoGluon, Microsoft NNI hides the technical complexity from users, who can easily use NNI with the same parameters. Nevertheless, with NNI, users need to define a specific hyper-parameter space that it will explore. All the information needs to be written in a `config.xml` file before running experiments.

In this work, we use AutoGluon and Microsoft NNI to perform optimization of hyper-parameters of Light Gradient Boosted Machine (LGBM). We set `nthreads_per_trial` to 96 (as we use AWS instances with 96 cores). We use AUROC (Area under the curve of Receiver Operating System) as the `evaluation metric` and Bayesian Optimization search for the search strategy. In the case of NNI, we use Tree-structured Parzen Estimator (TPE) as `Tuner`, and we set the number of parallel runs to 11, and the maximum number of trials to 99999, `time_budget` to 700 hours for NNI configuration. The objective of `Tuner` in NNI to achieve the maximum value of the true positive rate at a fixed false positive rate (e.g., 0.1%). As mentioned earlier, we define the search space for NNI with respect to the EMBER data set in Table 1 and SecureAge data set in Table 2.

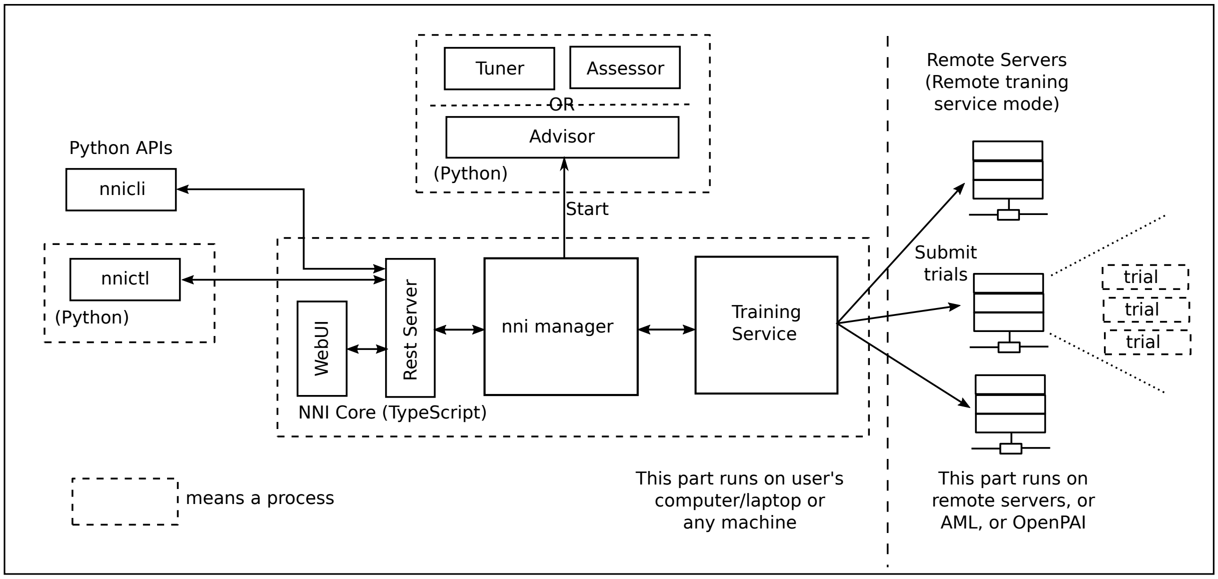

Figure 2: The high-level architecture of Microsoft NNI.

Table 1: Search Space of NNI used for EMBER data set

Parameter Name	Minimum Value	Maximum Value
n_estimators	2000	8000
learning_rate	0.02	0.6
num_leaves	40	4096
feature_fraction	0.50	1.00
bagging_fraction	0.50	1.00
min_data_in_leaf	4	65

Table 2: Search Space of NNI used for SecureAge data set

Parameter Name	Minimum Value	Maximum Value
n_estimators	50	4000
learning_rate	0.03	0.40
num_leaves	20	256
feature_fraction	0.80	1.00
bagging_fraction	0.70	1.00
min_data_in_leaf	3	50

4 EXPERIMENTS

We now present the experiments and analysis of results. We first present the data sets used in our work and then describe the experiments along with their analysis.

4.1 Malware Data Sets

4.1.1 EMBER Data Set. Working in malware detection is challenging due to the unavailability of a benchmark data set. The binary files often contain sensitive information such as private intellectual properties, personally identifiable information, sensitive network infrastructure information, network infrastructure details, etc. We use a publicly available data set named EMBER (Endgame Malware BEnchmark for Research) [1]. The authors of the data set collected samples within 2 years (2017 and 2018), each containing 1 million samples. In this work, we use the data set collected in 2018. Rather than providing raw binary files, this data set provides the features extracted from 1 million binary files including 800K training samples (300K malicious samples, 300K benign samples, and 200K unlabeled samples), and 200K test samples (100K malicious samples and 100K benign samples). To train a ML model, we do not use any unlabeled samples. The features and metadata are extracted from PE files using the LIEF parser. The authors of the data set also provided a benchmark LightGBM model that is trained on the vectorized features.

4.1.2 SecureAge Data Set. This is a private data set that includes raw binary files acquired from a company named SecureAge Technology. We used a subset of the data that includes 100K benign samples and 100K malicious samples. We use the feature extractor that was used for the EMBER data set to extract 2381 features for each binary file. We randomly select 90% of benign and malware PE files as the training set and the rest 10% as the test set. It is to be noted that the samples from training and test set are mutually exclusive, i.e., one PE file is either in training or test set. To generalize the capability of AutoML frameworks, we used LightGBM for training and testing on this data set.

4.2 Baseline Performance with Default Hyper-parameters

In [2], the authors reported the performance of LightGBM on the EMBER data set. At 0.1% of false positive rate (FPR), the reported true positive rate is 86.8%. This performance is obtained with the set of hyper-parameters presented in Table 3. The corresponding AUROC curve of the LightGBM classifier is shown in Figure 3. Here boosting type “gbdt” indicates the LightGBM classifier. The

Table 3: Hyper-parameters of LightGBM for Baseline performance on two Data Sets

Parameter	EMBER	SecureAge
boosting	gbdt	gbdt
objective	binary	binary
num_boost_round	10000	10000
early_stopping_rounds	10	10
n_estimators	1000	100
learning_rate	0.5	0.1
num_leaves	2048	31
feature_fraction	0.5	1.0
bagging_fraction	1.0	1.0
min_data_in_leaf	50	20

number of leaves (num_leaves) indicates the leaf nodes of LGBM. min_data_in_leaf indicates the minimum data in the leaf node of LightGBM. This parameter regulates the over-fitting of the classifier.

We carried out similar experiments on the SecureAge data set. The hyper-parameter setting of LightGBM used for this data set is presented in Table 3. We achieved 80.8% of TPR at 0.1% of FPR. The corresponding AUROC is shown in Figure 3. It is to be noted that with default hyper-parameters, LightGBM achieves very high TPR when FPR is greater than 0.1% for both data sets. For instance, it achieves 99.99% of TPR at 1% of FPR. However, this affects the practicality of the model as it requires significant users' effort to verify the samples that are actually false positive.

4.3 Efficacy of AutoML Techniques

To demonstrate the impact of AutoML techniques on the performance of LightGBM in malware detection, we first used AutoML techniques to fine-tune hyper-parameters of the LightGBM classifier before testing on the test sets.

4.3.1 Performance on EMBER Data Set. For both platforms, we used 10% of the training data as a validation set. In other words, we trained the LightGBM model with 90% of the data that belongs to the original EMBER training set. We used the AUROC score to evaluate the validation results and set early stopping criteria to 10 training iterations. In Table 4, we present the hyper-parameters of LightGBM after fine-tuned by AutoGluon on the EMBER data set. Using this set of hyper-parameters, the trained LightGBM achieves 90% of TPR at 0.1% of FPR. This means that the fine-tuned LightGBM outperforms the baseline model by increasing the TPR by 3.2%. Given 1000 samples to be analyzed, the fine-tuned LightGBM model correctly detects 32 malware samples more while raising only 1 false positive. This is a significant improvement as the number of samples analyzed every day by a malware detection system is in the order of a few hundreds of thousands.

Not only being able to fine-tune a specific model, AutoGluon also has an option of ensemble framework. It uses a weighted combination of multiple base models namely neural network, LightGBM, CatBoost, Random Forest, Extremely-randomized Trees, and K-Nearest Neighbors. The weights and the hyper-parameters of base models are learned in the framework. The fine-tuned ensemble framework also achieves 90% of TPR @ 0.1% of FPR. This is similar to the performance of the LightGBM model alone. The corresponding ROC curve of the ensemble of classifiers is shown in Figure 3. Comparing the ROC curves in Figure 3, it is evident that the ensemble of classifiers produces a more stable performance in terms of TPR with respect to the low value of FPR.

Table 4: Hyper-parameters values of LightGBM fine-tuned by AutoGluon on two Data Sets

Parameter	EMBER	SecureAge
boosting	gbdt	gbdt
objective	binary	binary
boosting_iterations	10000	10000
learning_rate	0.0390167	0.03766
num_leaves	66	81
feature_fraction	0.89714	0.7645
bagging_fraction	1.0	1.0
min_data_in_leaf	6	4

We achieved similar performance when using Microsoft NNI for fine-tuning hyper-parameters of LightGBM on the EMBER data set. Using the hyper-parameter set optimized by NNI, the LightGBM classifier achieves 89% of TPR when FPR is fixed at 0.1%. This corresponds to an improvement of 2.2% compared to the performance when using default hyper-parameters. Unlike AutoGluon, users could choose from different sets of hyper-parameters that result in a similar performance to the LightGBM classifier at the NNI framework. A trail represents a set of hyper-parameters at this framework. In Table 5, we present such three best hyper-parameter settings of the LightGBM Classifier optimized by NNI. The corresponding ROC curve of the LightGBM classifier using the first set of hyper-parameters is shown in Figure 3. The comparable performance of the LightGBM models fine-tuned by AutoGluon and NNI demonstrates that using an AutoML framework to optimize model hyper-parameters has a great impact on the classifier performance while hiding all technical complexity of ML models from non-expertise ML practitioners.

4.3.2 Performance on SecureAge Data Set. In this section, we evaluate the impact of AutoGluon and NNI on the performance of the LightGBM classifier when testing on the SecureAge data set. It is worth mentioning again that the LightGBM classifier achieves only 80.8% of TPR when the FPR is fixed at 0.1%. Similar to the experiments on the EMBER data set, we used 10% of the training data as a validation set. We also used the AUROC score to evaluate the validation result and set early stopping criteria to 10 iterations.

The LightGBM classifier after being fine-tuned by AutoGluon achieves 87.4% of TPR at 0.1% FPR. This corresponds to an improvement of 6.6% compared to the baseline performance. We note that unlike the EMBER data set that is a benchmarking but old data set, the SecureAge data set contains samples that are recently collected (i.e., in 2019 and 2020). Thus, it contains more sophisticated samples that have been enhanced by the malware authors to evade detection systems. This improvement is therefore more significant when applying to practical scenarios. The best hyper-parameter setting of the LightGBM classifier fine-tuned by AutoGluon is presented in Table 4 and the corresponding ROC curve is shown in Figure 4.

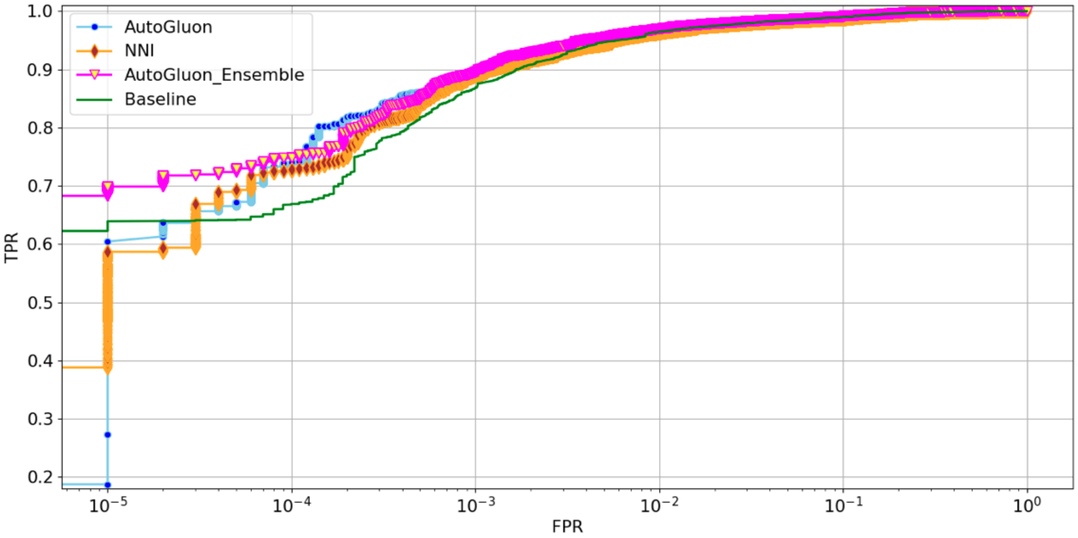

Figure 3: AUROC of LightGBM Fine-tuned by AutoGluon and NNI on EMBER Data Set compared with Baseline performance.

The LightGBM model after being fine-tuned by NNI also significantly improves the performance. TPR is increased to 85.4% when FPR is fixed at 0.1%. This corresponds to an improvement of 4.6%. The three best hyper-parameter settings of the LightGBM classifier optimized by NNI on the SecureAge data set are presented in Table 5. The corresponding ROC curve of the model using the first set of hyper-parameters is shown in Figure 4.

The comparison of time (in hrs) and physical RAM memory consumption (in GB) between AutoGluon and NNI for LGBM classifier on two Data Sets are presented in Table 6. Time and resource consumption depends on how many parallel threads are running and the parameters used by the model. We present the performance of the fine-tuned LightGBM model on test data that are split in time with respect to training data in Figure 5. We trained the models with EMBER training data and test them using SecureAge data. These two datasets were generated in diff rent time frame: EMBER in 2018 and SecureAge in 2020. Still, the models fine-tuned by the AutoML platform better with respect to the Baseline model especially at the low value of FPR. We used the LightGBM as Baseline model here, which was used as a Baseline model for EMBER data.

In summary, using two AutoML frameworks to fine-tune the models on two different data sets yields similar performance trends: improving the model performance compared to the cases without hyper-parameter optimization and also split data in time environment as EMBER and SecureAge data were generated in separate time frame. This demonstrates the generalized efficacy of AutoML techniques in helping non-expert ML practitioners fine-tune the hyper-parameters of ML models without the need for domain-specific knowledge. This is actually very useful for ML practitioners as well as end-users in the era that witnessed the proliferation of data analytics in any discipline.

Table 5: Hyper-parameters values of LightGBM fine-tuned by NNI on two Data Sets

Set Index	Parameter	EMBER	SecureAge
1	boosting	gbdt	gbdt
	objective	binary	binary
	n_estimators	5000	3000
	learning_rate	0.035	0.0390167
	num_leaves	72	81
	feature_fraction	0.8714	0.7
	bagging_fraction	0.8	1.0
	min_data_in_leaf	8	6
2	boosting	gbdt	gbdt
	objective	binary	binary
	n_estimators	3800	3800
	learning_rate	0.0390167	0.0390167
	num_leaves	66	81
	feature_fraction	0.89	0.7
	bagging_fraction	1.0	1.0
	min_data_in_leaf	7	6
3	boosting	gbdt	gbdt
	objective	binary	binary
	n_estimators	3900	2800
	learning_rate	0.0390167	0.0390167
	num_leaves	66	81
	feature_fraction	0.89	0.7
	bagging_fraction	0.80	1.0
	min_data_in_leaf	7	6

4.4 Cost-performance Trade-off

Despite the efficacy of AutoML techniques, a challenging issue while using them is the requirement of large-scale computational and storage resources. Due to limited resources in our lab, we run our experiments in Amazon web Service (AWS) environment. We used one instance with 96 cores and 192GB RAM (c5.24xlarge) and another instance with 72 cores 192GB RAM (c5n.18xlarge) for around 10 days. These two instances cost US$4.704 and US$4.464 per hour. To complete all the experiments on the EMBER data set

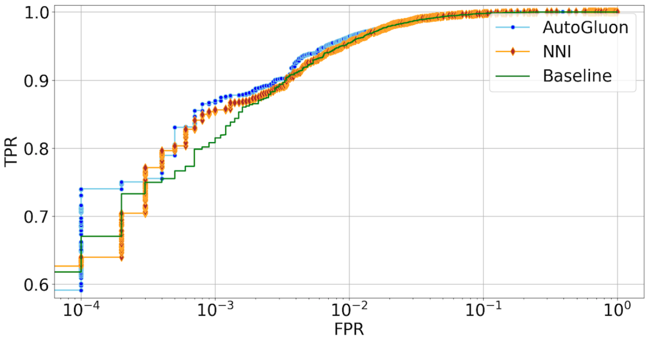

Figure 4: AUROC of LightGBM Fine-tuned by AutoGluon and NNI on SecureAge Data Set compared with Baseline performance.

Table 6: Comparison of time and resource consumption between AutoGluon and NNI for LightGBM on two Data Sets

Platforms	EMBER		SecureAge	
	Time (hrs)	RAM (GB)	Time (hrs)	RAM (GB)
AutoGluon	24	192	24	64
NNI	120	192	120	64

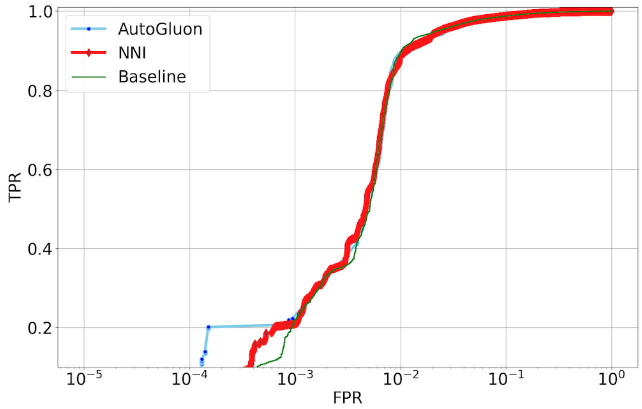

Figure 5: AUROC of LightGBM Fine-tuned by AutoGluon and NNI using EMBER training data and test on SecureAge Data Set and compared them with Baseline performance.

for both AutoGluon and NNI, we have spent in total US$5000.00. We also observe that to run the ensemble framework of AutoGluon, large-scale storage space is required to store all intermediate models produced by it. The experiments on the EMBER data set consumed 4 TB hard disk space in AWS S3. As SecureAge is private data set binding with the usage policy of the company, we could not run the experiments in the AWS environment. We used our in-house workstation with 20 cores 64GB RAM. Due to the storage resource constraint, we were not able to run the experiments with the ensemble framework with the SecureAge data set. Therefore, the trade-off between the performance and the cost needs to be taken into account while using AutoML techniques.

5 CONCLUSION

In this paper, we presented a comprehensive empirical evaluation of AutoML frameworks on hyper-parameter optimization. We considered two AutoML frameworks including AutoGluon and Microsoft NNI to optimize hyper-parameters of a LightGBM model for malware detection. We carried out experiments with two malware data sets including a publicly available benchmarking data set and a private data set which includes very recent malware instances. The experimental results show that AutoML techniques have a great impact on the performance of ML models by optimizing the hyper-parameters with respect to these data sets. After being fine-tuned by the AutoML frameworks, the performance of ML models is significantly improved compared to the baseline performance obtained with the with the known best performing or default parameter setting. Moreover, AutoML frameworks hide technical complexities from ML practitioners, thus the use of ML become more elegant to them, especially for non-expert ML users. Using a few lines of codes, users can integrate their ML models to those AutoML frameworks for fine-tuning and repetitive experiments. Through the empirical analysis, we also highlighted the trade-off between performance and cost of using AutoML frameworks as they require a large amount of computational and storage resources to run the optimization of hyper-parameters of ML models.

REFERENCES

[1] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *CoRR* abs/1804.04637 (Sept. 2018).

[2] Hyrum S. Anderson and Phil Roth. 2020. *The baseline performance of EMBER2018 on LGBM classifier.* Retrieved 2020-12-28 from https://docs.google.com/presentation/d/1A13tsUkgWeujTy9SD-vDFfQp9fnIqbSE_tCihNPlArQ/mobilepresent?slide=id.g6318784c2c_0_1088

[3] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proc. 1st Workshop on Deep Learning for Recommender Systems.* Boston, USA, 7–10.

[4] C. Chung, C. Chen, W. Shih, T. Lin, R. Yeh, and I. Wang. 2017. Automated machine learning for Internet of Things. In *IEEE ICCE-TW 2017.* 295–296.

[5] O. E. David and N. S. Netanyahu. 2015. DeepSign: Deep learning for automatic malware signature generation and classification. In *2015 International Joint Conference on Neural Networks (IJCNN).* 1–8.

[6] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, M. Li, and Alex Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. In *7th ICML Workshop on Automated Machine Learning (2020).*

[7] D. Gavrilut, M. Cimpoesu, D. Anton, and L. Ciortuz. 2009. Malware detection using machine learning. In *2009 International Multiconference on Computer Science and Information Technology.* 735–741.

[8] Cheng Guo and Felix Berkhahn. 2016. Entity Embeddings of Categorical Variables. arXiv:1604.06737 [cs.LG]

[9] Jeremy Howard and Sylvain Gugger. 2020. Fastai: A Layered API for Deep Learning. *Information* 11, 2 (Feb. 2020), 108.

[10] Sai Praveen Kadiyala, Akella Kartheek, and Tram Truong-Huu. 2020. Program Behavior Analysis and Clustering using Performance Counters. In *Proc. 2020 DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS) Workshop.* Virtual Event.

[11] Liu Liu, Bao sheng Wang, Bo Yu, and Qiu xi Zhong. 2017. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering* 18 (2017), 1336–1347.

[12] Microsoft. 2020. *Neural Network Intelligence (NNI).*

[13] Dima Rabadi and Sin G. Teo. 2020. Advanced Windows Methods on Malware Detection and Classification. In *Proc. Annual Computer Security Applications Conference (ACSAC '20).* Austin, USA, 54–68.

[14] Wee Ling Tan and Tram Truong-Huu. 2020. Enhancing Robustness of Malware Detection using Synthetically-adversarial Samples. In *Proc. IEEE Globecom 2020.* Taipei, Taiwan, 1–6.

[15] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine* 104 (2020), 101822.