# Bayesian Hyper-Parameter Optimisation for Malware Detection

Fahad T ALGorain[1], John A Clark[1]

[1]University of Sheffield, Sheffield S10 2TN, UK

**Abstract**

Malware detection is a major security concern and a great deal of academic and commercial research and development is directed at it. Machine Learning is a natural technology to address malware detection and many researchers have investigated its use. However, drawing comparisons between different techniques is a fraught affair. For example, the performance of ML algorithms often depends significantly on parametric choices, so the question arises as to what parameter choices are optimal. In this paper, we investigate the use of a variety of ML algorithms for building malware classifiers and also how best to tune the parameters of those algorithms – generally known as hyper-parameter optimisation. We examine the effects of some simple (model-free) ways of parameter tuning together with a state-of-the-art Bayesian model-building approach. Our work is carried out using EMBER, a major published malware benchmark dataset on Windows Portable Execution (PE) metadata samples.

**Keywords**

Hyper-parameter optimisation, Automated Machine Learning, Static Malware Detection, Tree Parzen Estimators, Bayesian optimisation, Random Search, Grid Search

## 1. Introduction

Malware is one of the most pressing problems in modern cybersecurity and its detection has been a longstanding focus for both academic and commercial research and development [1]. Detection must be effective (low false positives and negatives) but also efficient, particularly in areas such as forensics or threat hunting where vast file storage may need to be scanned for malware. Furthermore, the malware environment constantly changes and so (re-)training speed of detectors is also important [2]. Machine Learning (ML) is an obvious avenue to pursue, with various advantages to harnessing it: an ML approach can significantly reduce manual effort in developing detectors, giving more rapid deployment; it can play a critical role in the extraction of *insight* from malware samples; and ML-based detectors can detect *some* unseen malware, e.g. unseen malware that has features that are similar to those of known malware may be detected because of the loose pattern matching that underpins many ML classification approaches. A large number of ML techniques have been brought to bear on the malware detection problem, often attaining good results. However, ML must not be seen as a toolkit that can simply be thrown at a problem. Many ML techniques are parametrised and the choice of parameters may make a significant difference to performance. In modern, widely used ML

*AI-CyberSec 2021: Workshop on Artificial Intelligence and Cyber Security, December 14, 2021, Cambridge, UK*
ftalgorain1@sheffield.ac.uk (F. T. ALGorain); john.clark@sheffield.ac.uk (J. A. Clark)
0000-0003-0547-1402 (F. T. ALGorain); 0000-0002-9230-9739 (J. A. Clark)

toolkits, ML algorithms often have many tens of parameters (and sometimes more). This leads to the thorny issue of how such parameters may be best set, a problem generally referred to as *hyper-parameter optimisation* (HPO). HPO is a significant focus of research in optimisation and has the potential to improve on the results obtained by a specific detection approach, but also to enable fair comparison of techniques with techniques that are not the specific focus of investigation: comparing a new technique with 'vanilla' variants of existing techniques is a recurring motif in many studies. Bergstra et al. (2015) [3] advocate that hyper-parametrisation should be a 'formal outer loop' in the learning process, a view we very much support. Manual tuning is often simply impossible. (As an aside, we observe that many commercial users of ML spend a great deal of time tuning for their specific needs.) ML toolkits seek to address this problem by adopting *default values* for parameters; these values are set to values that have been shown to work *plausibly well* over many problems. However, for any specific problem at hand it is far from clear that the default values will be the best, or even good, choices. We have **significant domain incentives** to gain the best possible results for malware detection. Both false positive (FP) and false negative (FN) classifications have major consequences. The former lead to significant wasted effort investigating the (non-) malware together with denial of service to the application concerned, while the latter means malware goes undetected, with potentially catastrophic consequences. **For malware detection based on ML making high-performing hyper-parameter choices therefore matters.** In this paper we, initially explore a variety of ML techniques applied to classification of a specific form of malware (Windows PE files). We aim to demonstrate that hyper-parameter optimisation may have a significant effect on detection of this type of malware. More generally, we argue that it should play an important part in ML-based malware detection research research, and in security applications more widely. The contributions of our paper are:

1. A demonstration of how well various ML-based Windows Portable Executable (PE) file classifiers perform when trained with default parameters.
2. An evaluation of various hyper-parameter optimisation approaches applied to this problem, including:
   a) Established model-free techniques, e.g. Grid Search and Random Search; and
   b) A Bayesian optimisation model-based approach.
3. A demonstration that for our target problem the optimal choices of ML hyper-parameters may *vary considerably from the toolkit defaults.*

Windows PE files are an important malware vector, and their detection has been the focus of significant research. The work uses the EMBER dataset [4] – a recently published dataset comprising header and derived information from a million examples of PE files. This dataset contains examples of malware, benign software, and software of unknown malicious status. The dataset elements are labelled accordingly and so enable supervised learning. EMBER is now a major resource for ML and the malware community. Our evaluation includes both functional performance and efficiency (time to train). The last point above should not be underestimated. In a great many cases results will simply be sub-optimal if tuning is manual and the complexity of the parameter space is huge (and for many techniques this will be the case), and, as we have argued above, the costs of sub-optimality may be considerable.

## 2. Related Literature

### 2.1. Windows PE and ML Background

Numerous works have been produced regarding ML-based static Portable Executable (PE) malware detection such as [5][6][7], but work has often been hampered by the absence of a standard benchmark dataset. The publication of the EMBER dataset [4] has resolved this problem. A baseline model for supervised ML has been provided by the authors to aid in using the dataset as a benchmark. Their dataset is accompanied by Python access routines. In [8] the authors considered imbalanced data set issues and model training duration (by reducing feature dimensions). They also applied a static detection method using a Gradient Boosting Decision Tree Algorithm. Their model achieved better performance than the baseline model with less training time. (They used feature reduction based on the recommendation of the authors in [4].) Another approach utilized a subset of the EMBER dataset for their work and compared different ML models [9]. Their work is mainly concerned with scalability and efficiency. Their goal was to identify malware families. The proposed Random Forest model achieved a slightly better performance than the baseline model.

### 2.2. HPO Related literature

Multiple works have asserted the potential of hyper-parameter optimisation. For example [10] indicated the importance of parameter tuning for increasing accuracy, indicating that Random Search works better than Grid Search when tuning neural networks. Also, [11] applied standard tuning techniques to a decision tree on 102 datasets and calculated the accuracy differences between tuned and traditional models. [12][13] are concerned with the greedy forward search which seeks to identify the most important hyperparametter to change next. [14] stressed the importance of single hyperparameters after using sequential model-based optimisation (SMBO) tuning. ANOVA is used to measure hyperparameter importance. The authors in [15][16] assessed the performance of hyperparameters across different datasets. [16] also utilized surrogate models that allow setting the arbitrary hyperparameter configurations based on a limit on the number of evaluations carried out. Bayesian optimisation-based (BO) hyper-parameter search was used in [15][17] for speeding up the analysis of the work. The literature reveals that HPO, and in particular Bayesian optimisation-based (BO) approaches, have much to offer.

In this paper, we will use Automated Hyper-parameter Optimisation using Tree Parzen Estimators (AHBO-TPE) which takes advantage of BO to speed up the analysis of the search as well.

## 3. Hyper-parameter optimisation (HPO)

### 3.1. Definition and Motivation

Hyper-parameters are parameters of a model that are not updated during the learning process [17]. The HPO problem is defined similarly by many researchers as a search to find $x^*$

$$x^* = \arg\min_{x \in X} f(x), \tag{1}$$

where $f(x)$ is an objective score to be minimised. Commonly it is an error rate of some form evaluated on the validation set, e.g. the Root Mean Square Error (RMSE). $x^*$ is the hyper-parameter vector that gives rise to the lowest score value, and $x$ can be any vector of parameters in the specified domain. HPO seeks the hyper-parameter values that return the lowest score. For malware and similar classification tasks suitable choices for the objective function are holdout and cross-validation error. Furthermore, if we consider a loss function for the same problem a possible choice is a misclassification rate [18]. For our proposed model, the loss function is taken to be (ROC AUC score with cross-validation − 1). For in-depth background about validation protocols see [19]. Our work aims also to investigate evaluation time. There are two clear ways to do this. The first is to use a subset of folds in testing an ML algorithm [20]. The second is to use a subset of the dataset, especially if we have a big data set [21][22], or use a less iteration.

Although HPO has a great deal to offer, it comes at a price, in particular a computational price. For every hyper-parameter evaluation, we must train the model, make predictions on the validation set, and then calculate the validation metrics. Developing a robust ML-based classifier for Windows PE with a credibly sized and diverse dataset such as EMBER is therefore a significant undertaking. The computational costs involved act as a disincentive to implementing Bergstra et al.'s formal outer loop. There is a pressing need for traversing the hyper-parameter space efficiently and we demonstrate how a leading HPO approach allows us to do so in this paper.

Here, Windows PE files are a means to an end; the same issues apply to detecting other malware. Although malware is our major interest, our work also seeks to motivate consideration of HPO more widely in the application of ML in cybersecurity (since HPO issues apply there too). For more information about the HPO problem interested readers should refer to [18].

### 3.2. Model-free Blackbox optimisation methods (Search Methods)

Perhaps the two most common HPO methods are Grid Search and Random Search. These require only an evaluation function to work, i.e. they are what is commonly referred to as 'blackbox'. In Grid Search the individual parameters are discretised, i.e. a number of specific values are selected as 'covering' the particular parameter space. For example the elements in the set $\{0.0, 0.25, 0.5, 0.75, 1.0\}$ could be taken to cover a continuous parameter in the range [0.0..1.0]. Grid Search evaluates the function over the cross product of the discretised sets of hyper-parameters.

Random Search selects values randomly from the domain of each hyper-parameter. Usually, the values selected from each domain by Random Search are independent, i.e. the value of one parameter choice does not affect the value selected for another parameter choice. Furthermore, for an individual parameter all values generally have the same probability of being selected. It is possible to relax such properties, producing what is often referred to as a *biased* stochastic search. Such biases often encode for domain insight which is not in the spirit of a blackbox approach. In our work, we adopted a standard or 'vanilla' Random Search. Grid Search suffers from the 'curse of dimensionality' [23] . As the number of parameters increases or finer grain discretisation is adopted the computational complexity mushrooms. Furthermore, it is does not learn from past evaluations; we generally refer to such approaches as being *uninformed*. Consequently, it may spend a great deal of time evaluating candidates in regions where previous evaluation of candidates has given rise to poor objective values. Random Search will search the specified space until a certain number of evaluations, time, or budget has been reached. It works better than Grid Search when we know the promising hyper-parameter regions and so we can constrain the stochastic selection of candidates to lie somewhere in such regions [10][11]. Combining Random Search with complex strategies allows a minimum convergence rate and adds exploration that can improve model-based search [18][24]. Random Search is also an uninformed method and so takes a long time to identify the best performing hyper-parameter settings. It is not surprising that uninformed methods can be outperformed by methods that use evaluation history to judge where to try next; indeed, such guided search usually outperforms uninformed methods [15][25][26].

## 3.3. Bayesian optimisation (BO)

BO has emerged recently as one of the most promising optimisation methods for expensive blackbox functions. It has gained a lot of traction in the HPO community with significant results in several areas such as image classification, speech recognition, neural language modeling, For an in-depth preview about BO, the reader is referred to [17][27]. BO is an informative method that takes into consideration past results to find the best hyper-parameters. It utilizes those previous results to form a probabilistic model that is based on a probability of the score given a hyper-parameter which is denoted by the formula: $P(score|hyperparameter)$. [28] refers to the probabilistic model as a *surrogate* for the objective function denoted by $P(y|x)$, the probability of $y$ given $x$. The model or surrogate is more straightforward to optimize than the objective function. BO works to find the next hyper-parameters to be evaluated using the actual objective function by selecting the best performing hyper-parameters on the surrogate function. A 5-step processes to do this is given by [28]. The first step builds a surrogate probability model of the objective function. The second finds the hyper-parameters with best results on the surrogate. The third applies those values to the real defined objective function. The fourth updates the surrogate with this new real objective function result. Steps 2–4 are repeated until the maximum iteration or budgeted time is reached [29]. BO has two primary components: probabilistic model and an acquisition function to decide the next place to evaluate. Furthermore, BO trades off exploration and exploitation; instead of assessing the costly blackbox function, the acquisition function is cheaply computed and optimized. There are many choices for the acquisition function but here we use the most common – expected improvement (EI) [30]. The goal of Bayesian reasoning is to become more accurate as more performance data is acquired. The previous 5-step processes is repeated to keep the surrogate model updated after each evaluation of the objective function [15]. BO spends a little more time generating sets of hyper-parameter choices that are likely to provide real improvements whilst keeping calls to the actual objective function as low as possible. Practically, the time spent on choosing the next hyper-parameters to evaluate is often trivial compared to the time spent on the (real) objective function evaluation. BO can find better hyper-parameters than Random Search in fewer iterations [25]. This is one of the issues we seek to address in our work: whether AHBO-TPE could find better hyper-parameters than Random Search with fewer iterations specially for our target domain - Windows PE file malware detection.

## 3.4. Sequential Model-Based optimisation (SMBO)

There are several options for the SMBO's evaluation of the surrogate model $P(y|x)$ [15]. One of the choices is to use Expected Improvement (EI) as defined in the equation below:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)P(y|x)dy \tag{2}$$

Here $y^*$ is the threshold value of the objective function, $x$ is the hyper-parameter, $y$ is the actual value of the objective function using the hyper-parameters $x$, and $P(y|x)$ is the surrogate probability model expressing the probability (density) of $y$ given $x$. The goal is to find the best hyper-parameters under function $P(y|x)$. The threshold value $y^*$ is the best objective value obtained so far. We aim to improve on (i.e. get a lower value than) the best value obtained so far. For such minimisation problems, if a value $y$ is greater than the threshold value, then it is not an improvement. Only values less that the threshold are improvements. For a value $y$ less than the threshold $y^* - y$ is the improvement. By integrating over all such improvements, weighted by the density function $P(y|x)$ gives the overall expected improvement given the vector of parameter values $x$. When better values of $x$ are found (i.e. giving rise to actual improvements in the real objective function) the threshold value $y^*$ is updated. The above description is an *idealised* view of Expected Improvement. In practice the choice of threshold value is more flexible, i.e. $y^*$ need not be the best objective value witnessed so far; this is actually the case for the Tree-Parzen Estimator approach outlined immediately below.

## 3.5. Tree-Structured Parzen Estimators (TPE)

TPE constructs its model utilizing Bayesian rules. Its model $P(y|x)$ is built from two model components. One component models values less than a threshold and the other models values greater than that threshold.

$$P(x|y) = \begin{cases} l(x) \;\; if \;\; y < y^* \\ g(x) \;\; if \;\; y >= y^* \end{cases} \tag{3}$$

TPE uses $y^*$ to be some quantile $\gamma$ of the observed $y$ values, i.e. such that $P(y < y^*) = \gamma$ [3]. This allows data to be available to construct the indicated densities. $l(x)$ is a density based on the set of evaluated values of $x$ that have been found to give objective values less than the threshold. $g(x)$ is the density based on the remaining evaluated $x$ values. Here $P(x|y)$ is the density of hyper-parameter $x$ given a objective function score of $y$. It is expressed as follows [15]:

$$P(y|x) = \frac{P(x|y) * P(y)}{P(x)} \tag{4}$$

Bergstra et al. (2012) also show that to maximise improvement we should seek parameters $x$ with high probability under $l(x)$ and low probability under $g(x)$. Thus, they seek to maximise $g(x)/l(x)$. The best such $x$ outcome is then evaluated in the actual objective function and will be expected to have a better value. The surrogate model estimates the objective function; if the hyper-parameter that is selected does not make an improvement, the model won't be updated. The updates are based upon previous history/trials of the objective function evaluation. As mentioned before, the previous trials are stored in a pair of (score, hyper-parameters) by the algorithm after building the lower threshold density $l(x)$ and higher threshold density $g(x)$. It uses the history of these previous trials to improve the objective function with each iteration. The motivation to use TPE with SMBO to reduce time and find better hyper-parameters came from other papers [15][25][31][32]. SMBO uses Hyperopt [3], a Python library that implements BO or SMBO. Hyperopt makes SMBO an interchangeable component that could be applied to any search problem. Hyperopt supports more algorithms but we are interested in TPE only in our work. **Our contribution lies in the demonstration of the usefulness of SMBO using TPE for malware classification purposes.**

## 4. Experiments

Here we outline the experiments carried out and provide sample data and execution environment details. Discussion of results is given in the next section.

### 4.1. Execution Environment and Dataset

Our machine learning models evaluation used Scikit-learn [33] and Hyperopt [3]. The experiments were carried out on Windows 10 operating system, 8GB RAM, AMD Ryzen 5 3550 H with Radeon Vega Mobile Gfc 2.10 GHz, 64-bit operating system, x64-based processor. Also, a MacBook Air(Catalina version 10.15), 1.8 GHz Dual-core Intel i5, 8GB 1600 Mhz DDR3, Intel HD graphics 6000 1536MB. Version 2018 of the EMBER dataset [34] were used. It contains 1M samples in total. We used 300k benign and 300k malicious samples for training with 100k benign and 100k malicious samples for testing purposes. The 200k unlabelled examples of the dataset were not used in our experiments. Furthermore, the results were obtained using Jupyter Notebook version 6.1.0 and Python version 3.6.0.

### 4.2. Experiments with Default Settings

Table 1 shows the results when various ML techniques are applied with default parameter settings. The techniques include well-established approaches (Stochastic Gradient Descent classifier (SGD), Logistic Regression classifier (LR), Gaussian Naïve Bayes (GNB), K-nearest Neighbour (KNN), and Random Forest (RF) [33], [35]) and a state-of-the-art approach – Light-GBM [36]. This technique has over a hundred parameters and so introduces major challenges for hyper-parametrisation. Some of its categorical parameters (e.g. boosting type) give rise to conditional parameters. For initial experiments we adopted the default parameter settings adopted by the Scikit-Learn toolkit for all techniques other than LightGBM (which has its own defaults). The evaluation metric is **Area Under the Receiver Operating Characteristic Curve** (ROC AUC) [37]. ROC AUC plays an important role in many security classification tasks, e.g. it also occurs frequently as an evaluation metric in intrusion detection research. Grid Search results were obtained using the MacBook Air, while the rest (AHBO-TPE and Random Search) were obtained under the Windows 10 operating system for faster performance.

### 4.3. Hyper-parameter Optimisation

The most promising of the evaluated ML algorithms, taking into account functional performance and speed of training, was LightGBM. We choose to further explore hyper-parameter optimisation on this technique. Since LightGBM has over a 100 parameters, some of which are continuous, we simply cannot do exhaustive search. Accordingly, we have had to select parameters as a focus in this work. We focused on what we believe are the most important parameters. For Grid Search in particular we had to be particularly selective in what we optimised. Moreover, for Random Search we specified a budget of 100 iterations. We examine Grid Search, Random Search, and AHBO with Tree Parzen Estimators as HPO approaches. We therefore provide comparison between model-free (blackbox) approaches (Grid Search and Random Search), and AHBO-TPE, an approach that uses evaluation experience to continually update its model and suggest next values of the hyper-parameters. We applied AHBO-TPE in two phases, the first one we initially set it to 3 iterations, while the second being allowed 100 more iterations for fair comparison (with Random Search).

### 4.4. Additional Materials

Implementation details of our experiments can be found on our github repository [38].

## 5. Results and discussion

We have carried out our experiments on a benchmark dataset that has been assembled, in part, to include examples that present challenges to ML classification approaches [34]. The results given in Table 1 show that **the major ML approaches vary *hugely* in their suitability for the static malware classification tasks**. Our benchmark dataset is comprised of summaries of Windows PE files. It would seem prudent for malware detection researchers to evaluate multiple ML algorithms for non-Windows malware detection too. The results also show the clear promise of using a particular state-of-the-art algorithm – LightGBM – for the malware detection task. Taking both time and performance into account, Table 1 shows that LightGBM is clearly the best performing approach. The subsequent tables summarise our attempts to apply HPO approaches to the most promising of the original ML techniques. We can see in Table 2 that **HPO can offer significant improvements**. Random search performs very well. So does AHBO-TPE, but the initial optimisation is far more efficient. Figure 1 further illustrates how Random Search and AHBO-TPE evolve as iteration number increases. The tool default parameter choices cannot be relied upon to produce the best or even good results. Tables 5, 6 and 7 illustrate the difficulty of manually tuning parameters for this task. In some cases **the defaults and the best found values are at the opposite ends of the parameter ranges**, e.g. the bagging fraction in Table 5. Many are significantly different to the default value, e.g. $num\_leaves$ in Tables 6 and 7 and $n\_estimators$ of Table 7. Some binary choices are reversed, e.g. $objective$ and $is\_unbalanced$ of Table 6. Further results are given in the appendix were we use AHBO-TPE with the selected ML techniques. The results are shown with 10 iterations (a constraint imposed for reasons of computational practicality) and 3 fold cross validation. Moreover, the hyper-parameters that are promising in each ML model are added in Tables 7, 8, 9, 10 and 11. Also, in Figure 2 a comparison is given between our search methods and the benchmark model.

**Table 1**
Comparison Between Default ML Models Parameters Scores

| ML Model | Time to train | Score (AUC-ROC) |
|---|---|---|
| GNV | 11 min 56 s | 0.406 |
| SGD | 19 min 11s | 0.563 |
| LightGBM Benchmark | 26 min | 0.922 |
| RF | 57 min and 52s | 0.90 |
| LR | 1 hrs and 44 min | 0.598 |
| KNN | 3 hrs 14 min and 59s | 0.745 |

**Table 2**
Search Methods Score Comparison

| Search Methods | Best ROC AUC score | Number of Iteration | Time To Complete Search |
|---|---|---|---|
| BenchMark LightGBM Model non optimized | 0.922 | 100 | 26 mins |
| Grid Search optimisation | 0.944 | 965 | Almost 3 months |
| Random Search optimisation | 0.955 | 60 | 15 days, 13 hrs and 12 mins |
| AHBO-TPE initial Optimisation | 0.955 | 3 | 4 hrs |
| AHBO-TPE with extra 100 iteration optimisation | 0.957 | 26 | 27 days, 23 hrs and 11 mins |

**Table 3**
Selected ML Models with AHBO-TPE Score Comparison

| ML Model | Score (AUC-ROC) | Score (AUC-ROC) After optimisation |
|---|---|---|
| GNV | 0.406 | same |
| SGD | 0.563 | 0.597 |
| LightGBM Benchmark | 0.922 | 0.957 |
| RF | 0.901 | 0.936 |
| LR | 0.598 | 0.618 |
| KNN | 0.745 | 0.774 |

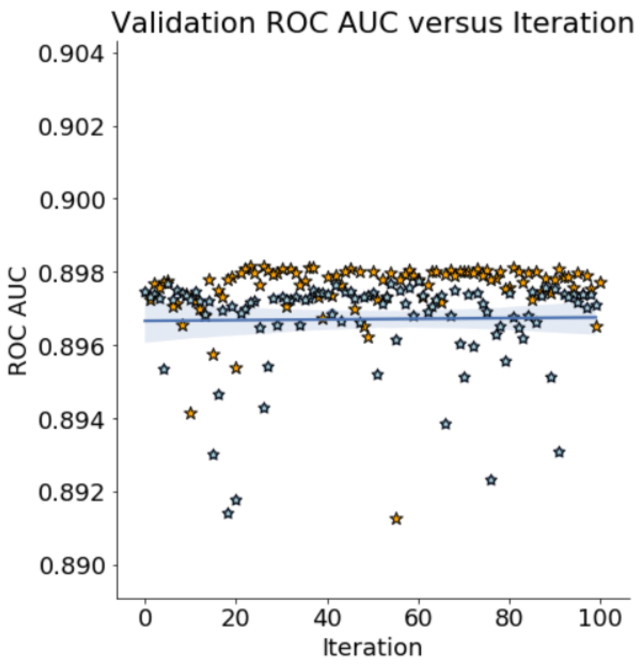

**Figure 1:** Highest Validation Score (the star is the best score achieved through iterations) AHBO-TPE (yellow) with Random Search Hyperparameter (blue) optimisation.

**Table 4**
Selected ML Models with AHBO-TPE Performance Results

| ML Model | Time to train | Training Time Reduction by |
|---|---|---|
| GNV | 11 min 56s | same |
| SGD | 4 min 35s | 14 min 35s |
| LightGBM Benchmark | 18 min 30s | 8mins |
| RF | 31 min 14s | 26 min |
| LR | 1 hr 5min 37s | 38 min |
| KNN | 4hrs 37 min and 30s | increased by 1hr 23min 29s |

## 6. Limitations and Conclusions

### 6.1. Limitations

A significant limitation is that the work uses a specific dataset (EMBER) concerned with Windows PE files. This limits rigorously founded generalisation of results. Evaluation using further malware datasets would build further insight into applicability beyond the direct case study malware type reported here. Also we have used a single (albeit highly effective) ‘informed’ hyper-parametrisation approach. The use of other informed hyper-parametrisation approaches would provide further insight and possible improvements. For practical purposes we informally identified plausible parameters that should be subject to variation and allowed the remaining ones to be set at the defaults. It is possible that improvements in results could be obtained by allowing variation in those parameters fixed at their default values.

### 6.2. Conclusions

The results overall would suggest that researchers in malware and ML are missing a significant opportunity to improve results attained by specific techniques of interest. The importance of sound optimisation in this domain is considerable: every improvement matters to the security of the system and there are major cost implications for gaining improvements. As, we argued earlier, sub-optimality may have major costs. Researchers who apply particular techniques to malware detection are interested in gaining maximal performance and showing their technique (accurately) at its best. We need a fair comparison of techniques at their best. Fair comparison is also crucial to the development of the field. HPO has the ability to place ML-based malware detection on a sound empirical footing. We recommend hyper-parameter optimisation for malware detection and the ML community. We also recommend Bayesian optimisation as a particularly promising approach with informative approaches generally being an important avenue for future research. For future work it seems investigating the applicability of our model in dynamic malware detection settings is worthwhile. The major available dataset (EMBER) contains unknown malware samples. Accordingly, applying HBO in a semi-supervised regime seems a plausible avenue to pursue.

We have shown how Bayesian Optimisation can be harnessed to advantage. We have applied a leading state-of-the-art approach to show how malware detection can be enhanced. We should be alert to major developments in the HPO field to ensure we deliver the very best hopes for malware detection stakeholders. In summary: the parametrisation of ML approaches exerts significant effect on the results obtained; Hyper-parameter Optimisation is an important device for obtaining best results for any specific technique; it is arguably an essential component for the rigorous comparison of techniques, which is a fundamental criterion for the development of the topic. It would be very useful for the ML-based malware detection community to agree comparison protocols. The above are our conclusions for ML applied to malware detection (here malicious Windows PE files). We propose that **HPO be an essential element of the ML process for malware detection applications**.

## References

[1] A. K. Pandey, A. K. Tripathi, G. Kapil, V. Singh, M. W. Khan, A. Agrawal, R. Kumar, R. A. Khan, Trends in malware attacks: Identification and mitigation strategies, in: Critical Concepts, Standards, and Techniques in Cyber Forensics, IGI Global, 2020, pp. 47–60.

[2] A. Al-Sabaawi, K. Al-Dulaimi, E. Foo, M. Alazab, Addressing malware attacks on connected and autonomous vehicles: Recent techniques and challenges, in: Malware Analysis Using Artificial Intelligence and Deep Learning, Springer, 2021, pp. 97–119.

[3] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D. D. Cox, Hyperopt: a python library for model selection and hyperparameter optimization, Computational Science & Discovery 8 (2015) 014008.

[4] H. S. Anderson, P. Roth, Ember: an open dataset for training static pe malware machine learning models, arXiv preprint arXiv:1804.04637 (2018).

[5] M. G. Schultz, E. Eskin, F. Zadok, S. J. Stolfo, Data mining methods for detection of new

malicious executables, in: Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001, IEEE, 2000, pp. 38–49.

[6] J. Z. Kolter, M. A. Maloof, Learning to detect and classify malicious executables in the wild., Journal of Machine Learning Research 7 (2006).

[7] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, C. K. Nicholas, Malware detection by eating a whole exe, in: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[8] H.-D. Pham, T. D. Le, T. N. Vu, Static pe malware detection using gradient boosting decision trees algorithm, in: International Conference on Future Data and Security Engineering, Springer, 2018, pp. 228–236.

[9] C. Fawcett, H. H. Hoos, Analysing differences between algorithm configurations through ablation, Journal of Heuristics 22 (2016) 431–458.

[10] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization., Journal of machine learning research 13 (2012).

[11] F. Hutter, H. Hoos, K. Leyton-Brown, An efficient approach for assessing hyperparameter importance, in: International conference on machine learning, PMLR, 2014, pp. 754–762.

[12] J. N. Van Rijn, F. Hutter, Hyperparameter importance across datasets, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2367–2376.

[13] A. Biedenkapp, M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, H. Hoos, Efficient parameter importance analysis via ablation with surrogates, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 31, 2017.

[14] K. Eggensperger, M. Lindauer, H. H. Hoos, F. Hutter, K. Leyton-Brown, Efficient benchmarking of algorithm configurators via model-based surrogates, Machine Learning 107 (2018) 15–41.

[15] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: 25th annual conference on neural information processing systems (NIPS 2011), volume 24, Neural Information Processing Systems Foundation, 2011.

[16] P. Probst, A.-L. Boulesteix, B. Bischl, Tunability: Importance of hyperparameters of machine learning algorithms., J. Mach. Learn. Res. 20 (2019) 1–32.

[17] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. De Freitas, Taking the human out of the loop: A review of bayesian optimization, Proceedings of the IEEE 104 (2015) 148–175.

[18] M. Feurer, F. Hutter, Hyperparameter optimization, in: Automated Machine Learning, Springer, Cham, 2019, pp. 3–33.

[19] B. Bischl, O. Mersmann, H. Trautmann, C. Weihs, Resampling methods for meta-model validation with recommendations for evolutionary computation, Evolutionary computation 20 (2012) 249–275.

[20] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown, Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013, pp. 847–855.

[21] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, et al., Fast bayesian hyperparameter optimization on large datasets, Electronic Journal of Statistics 11 (2017) 4945–4968.

[22] O. Maron, A. W. Moore, The racing algorithm: Model selection for lazy learners, Artificial

Intelligence Review 11 (1997) 193–225.
[23] R. Bellman, Dynamic programming princeton university press princeton, New Jersey Google Scholar (1957).
[24] F. Hutter, H. Hoos, K. Leyton-Brown, An evaluation of sequential model-based optimization for expensive blackbox functions, in: Proceedings of the 15th annual conference companion on Genetic and evolutionary computation, 2013, pp. 1209–1216.
[25] J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: International conference on machine learning, PMLR, 2013, pp. 115–123.
[26] S. Falkner, A. Klein, F. Hutter, Bohb: Robust and efficient hyperparameter optimization at scale, in: International Conference on Machine Learning, PMLR, 2018, pp. 1437–1446.
[27] E. Brochu, V. M. Cora, N. De Freitas, A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, arXiv preprint arXiv:1012.2599 (2010).
[28] I. Dewancker, M. McCourt, S. Clark, Bayesian optimization primer, 2015. URL: https://static.sigopt.com/b/20a144d208ef255d3b981ce419667ec25d8412e2/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf.
[29] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Auto-sklearn: efficient and robust automated machine learning, in: Automated Machine Learning, Springer, Cham, 2019, pp. 113–134.
[30] R. J. Donald, Efficient global optimization of expensive black-box function, J. Global Optim. 13 (1998) 455–492.
[31] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown, Towards an empirical foundation for assessing bayesian optimization of hyperparameters, in: NIPS workshop on Bayesian Optimization in Theory and Practice, volume 10, 2013, p. 3.
[32] L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316.
[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, the Journal of machine Learning research 12 (2011) 2825–2830.
[34] H. S. Anderson, P. Roth, elastic/ember, 2021. URL: https://github.com/elastic/ember/blob/master/README.md.
[35] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al., Api design for machine learning software: experiences from the scikit-learn project, arXiv preprint arXiv:1309.0238 (2013).
[36] Lightgbm documentation, ???? URL: https://lightgbm.readthedocs.io/en/latest.
[37] sklearn.metrics.roc_auc_score, ???? URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.
[38] F. ALGorain, J. Clark, Bayesian hyper parameter optimization for malware detection, 2021. URL: https://github.com/fahadgorain/Bayesian-Hyper-Parameter-Optimization-for-Malware-Detection.
[39] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, S. Anwar, Static malware detection and attribution in android byte-code through an end-to-end deep system, Future

Generation Computer Systems 102 (2020) 112–126.
[40] F. Cohen, Computer viruses: theory and experiments, Computers & security 6 (1987) 22–35.
[41] Willkoehrsen, Automated model tuning, 2018. URL: https://www.kaggle.com/willkoehrsen/automated-model-tuning.
[42] S.-H. Zhang, C.-C. Kuo, C.-S. Yang, Static pe malware type classification using machine learning techniques, in: 2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA), IEEE, 2019, pp. 81–86.
[43] R. G. Mantovani, T. Horváth, R. Cerri, J. Vanschoren, A. C. de Carvalho, Hyper-parameter tuning of a decision tree induction algorithm, in: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS), IEEE, 2016, pp. 37–42.
[44] F. Hutter, H. H. Hoos, K. Leyton-Brown, Identifying key algorithm parameters and instance features using forward selection, in: International Conference on Learning and Intelligent Optimization, Springer, 2013, pp. 364–381.
[45] V. Nguyen, S. Gupta, S. Rana, C. Li, S. Venkatesh, Filtering bayesian optimization approach in weakly specified search space, Knowledge and Information Systems 60 (2019) 385–413.
[46] B. Shahriari, A. Bouchard-Côté, N. Freitas, Unbounded bayesian optimization via regularization, in: Artificial intelligence and statistics, PMLR, 2016, pp. 1168–1176.
[47] H. J. Escalante, M. Montes, L. E. Sucar, Particle swarm model selection., Journal of Machine Learning Research 10 (2009).
[48] B. Komer, J. Bergstra, C. Eliasmith, Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn, in: ICML workshop on AutoML, volume 9, Citeseer, 2014, p. 50.
[49] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: International conference on learning and intelligent optimization, Springer, 2011, pp. 507–523.
[50] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, arXiv preprint arXiv:1206.2944 (2012).
[51] G. E. Dahl, T. N. Sainath, G. E. Hinton, Improving deep neural networks for lvcsr using rectified linear units and dropout, in: 2013 IEEE international conference on acoustics, speech and signal processing, IEEE, 2013, pp. 8609–8613.
[52] G. Melis, C. Dyer, P. Blunsom, On the state of the art of evaluation in neural language models, arXiv preprint arXiv:1707.05589 (2017).
[53] J. Singh, J. Singh, A survey on machine learning-based malware detection in executable files, Journal of Systems Architecture (2020) 101861.

## 7. Appendix

**Table 5**
LightGBM Grid Search Hyperparameter results

| Hyper-parameter | Grid Search Best Hyper-parameter Settings | Range | Default Value |
|---|---|---|---|
| boosting_type | GBDT | GBDT,DART,GOSS | GBDT |
| num_iteration | 1000 | 500:1000 | 100 |
| learning_rate | 0.005 | 0.005:0.05 | 0.1 |
| num_leaves | 512 | 512:2048 | 31 |
| feature_fraction(subsample) | 1.0 | 0.5:1.0 | 1.0 |
| bagging_fraction(subsample_freq) | 0.5 | 0.5:1.0 | 1.0 |
| objective | binary | binary | None |

**Table 6**
LightGBM Random Search Hyperparameter results

| Hyperparameter | Random Search Best Hyperparameteres | Range | Default Value |
|---|---|---|---|
| boosting_type | GBDT | GBDT or GOSS | GBDT |
| num_iteration | 60 | 1:100 | 100 |
| learning_rate | 0.0122281 | 0.005:0.05 | 0.1 |
| num_leaves | 150 | 1:512 | 31 |
| feature_fraction(subsample) | 0.8 | 0.5:1.0 | 1.0 |
| bagging_fraction(subsample_freq) | 0.8 | 0.5:1.0 | 1.0 |
| objective | binary | binary only | None |
| min_child_samples | 165 | 20:500 | 20 |
| reg_alpha | 0.102041 | 0.0:1.0 | 0.0 |
| reg_lambda | 0.632653 | 0.0:1.0 | 0.0 |
| colsample_bytree | 1.0 | 0.0:1.0 | 1.0 |
| subsample | 0.69697 | 0.5:1.0 | 1.0 |
| is_unbalance | True | True or False | False |

**Table 7**
LightGBM AHBO-TPE Search Hyper-parameter results

| Hyperparameter | AHBO-TPE Search Hyperparameter Results | Range | Default Value |
|---|---|---|---|
| boosting_type | GBDT | GBDT or GOSS | GBDT |
| num_iteration | 26 | 1:100 | 100 |
| learning_rate | 0.02469 | 0.005:0.05 | 0.1 |
| num_leaves | 229 | 1:512 | 31 |
| feature_fraction(subsample) | 0.78007 | 0.5:1.0 | 1.0 |
| bagging_fraction(subsample_freq) | 0.93541 | 0.5:1.0 | 1.0 |
| objective | binary | binary only | None |
| min_child_samples | 145 | 20:500 | 20 |
| reg_alpha | 0.98803 | 0.0:1.0 | 0.0 |
| reg_lambda | 0.45169 | 0.0:1.0 | 0.0 |
| colsample_bytree | 0.89595 | 0.0:1.0 | 1.0 |
| subsample | 0.63005 | 0.0:1.0 | 1.0 |
| is_unbalance | True | True or False | False |
| n_estimators | 1227 | 1:2000 | 100 |
| Subsample_for_bin | 160000 | 2000:200000 | 200000 |

**Table 8**
SGD Model AHBO-TPE Search Hyperparameter Results

| Hyperparameter | AHBO-TPE Search Hyperparameter Results | Range | Default Value |
|---|---|---|---|
| Penalty | L2 | L1,L2, elasticnet | L1 |
| Loss | Hinge | hinge, log, modified-huber, squared-hinge, | Hinge |
| Max-iterations | 10 | 10:200 | 1000 |

**Table 9**
RF Model AHBO-TPE Search Hyperparameter Results

| Hyperparameter | AHBO-TPE Search Hyperparameter Results | Range | Default Value |
|---|---|---|---|
| n_estimators | 100 | 10:100 | 10 |
| max_depth | 30 | 2:60 | None |
| max_features | auto | auto,log2,sqrt | auto |
| min_samples_split | 10 | 2:10 | 2 |
| min_samples_leaf | 30 | 1:10 | 1 |
| criterion | gini | gini,entropy | gini |

**Table 10**
LR Model AHBO-TPE Search HyperparameterResults

| Hyperparameter | AHBO-TPE Search Hyperparameter Results | Range | Default Value |
|---|---|---|---|
| max_iter | 200 | 10:200 | 100 |
| C | 8.0 | 0.0:20.0 | auto |
| solver | sag | 'liblinear','lbfgs', 'sag', 'saga' | lbfgs |

**Table 11**
KNN Model AHBO-TPE Search Hyperparameter Results

| Hyperparameter | AHBO-TPE Search Hyperparameter Results | Range | Default Value |
|---|---|---|---|
| n_neighbors | 15 | 1:31 | 5 |

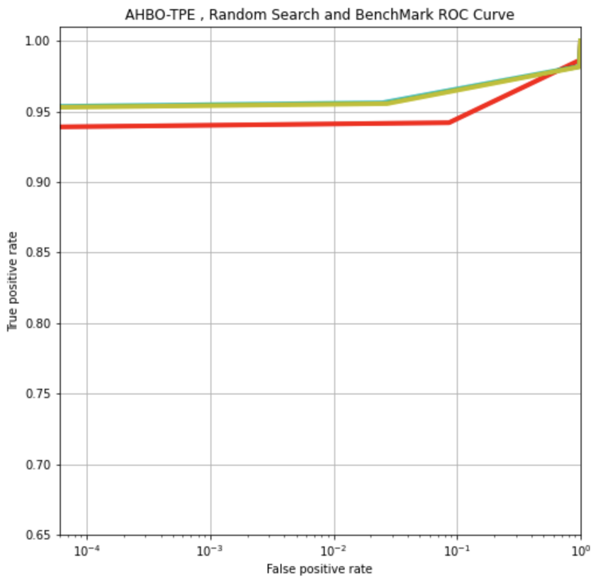

**Figure 2:** Shows ROC AUC Comparison for AHBO-TPE (Cyan color), Random Search (Yellow color) and Default Benchmark Model (Red color).