Small Effect Sizes in Malware Detection? Make Harder Train/Test Splits!

Tirth Patel[1], Fred Lu[1,2], Edward Raff[1,2], Charles Nicholas[1], Cynthia Matuszek[1] and James Holt[3]

[1]University of Maryland, Baltimore County, 1000 Hilltop Cir, Baltimore, MD 21250
[2]Booz Allen Hamilton, 8283 Greensboro Drive, McLean, VA 22102
[3]Laboratory for Physical Sciences, 5520 Research Park Drive, Catonsville, MD 21228

Abstract

Industry practitioners care about small improvements in malware detection accuracy because their models are deployed to hundreds of millions of machines, meaning a 0.1% change can cause an overwhelming number of false positives. However, academic research is often restrained to public datasets on the order of ten thousand samples and is too small to detect improvements that may be relevant to industry. Working within these constraints, we devise an approach to generate a benchmark of configurable difficulty from a pool of available samples. This is done by leveraging malware family information from tools like AVClass to construct training/test splits that have different generalization rates, as measured by a secondary model. Our experiments will demonstrate that using a less accurate secondary model with disparate features is effective at producing benchmarks for a more sophisticated target model that is under evaluation. We also ablate against alternative designs to show the need for our approach.

1. Introduction

Malware detection, determining if a given file is benign or malicious, is an important safety problem, since malware causes billions in financial damage each year [1]. However, it is not easy for academic researchers to know that they have produced an improvement using freely available data. This is because industry uses tens of millions of executables at tens of terabytes in scale to detect meaningful improvements in accuracy [2, 3, 4, 5]. In contrast, academic datasets with raw executables available are measured in tens of thousands of executables [6, 7, 8]. This small scale has made it easy for academic work to over-fit to the data [9, 10, 11], and best practices like a train and test set split by time (by when the executable was created) are not possible due to lack of information [11].

The goal of this work is to provide academic researchers with a means of constructing new train/test splits, using publicly available information for Microsoft windows malware, that can increase the predictive difficulty of the task by removing common biases that lead to overfitting. The crux of our method is that malware can be grouped into families of related type [12], and an ideal malware detector is one that can detect new families that were not seen during

CAMLIS'23: Conference on Applied Machine Learning for Information Security, October 19–20, 2023, Arlington, VA
tpatel9@umbc.edu (T. Patel); Lu_Fred@bah.com (F. Lu); Raff_Edward@bah.com (E. Raff); nicholas@umbc.edu (C. Nicholas); cmat@umbc.edu (C. Matuszek); holt@lps.umd.edu (J. Holt)

training. This insight gives us an objective way to group samples into train/test splits that do not cause significant information leakage by having the same malware families in both training and testing, as some prior academic works do [11]. By searching for malware families of the right difficulty to place in each train and test split, we can produce new benchmark splits for researchers to use that are smaller than the source datasets, but avoid the bias problems mentioned above.

The rest of our paper is organized as follows. In section 2 we will discuss the important related work to our own, including prior issues in malware detection research and the work in reproducibility and model selection that can be better leveraged by our benchmarks. Then in section 3 we will describe how we use a base, simpler model with a search procedure to construct these benchmark datasets. The goal is that our splits will have a lower baseline accuracy for existing methods, showing that we can produce a harder dataset, which in turn makes it easier to detect improvements in generalization and thus effect size. We demonstrate this for three difficulty levels (Easy, Medium, and Hard) in section 4, and that two intuitive ablation strategies are ineffective in subsection 4.1. Finally, our article concludes in section 5.

2. Related Work

Malware detection research using machine learning has been active since 1995 [13], and includes raw byte [14], API calls and assembly [15, 16], graph [17], or exogenous metadata [18]. However, much industrial research has indicated that academic methods do not often transfer well to industrial data, and so increasingly industry is trying to release more representative datasets [19, 2]. Such efforts are commendable, but these datasets often still require a VirusTotal license[1] to get the original files, and they can be prohibitively large. The SOREL-20M corpus has over 20 million files in a train/validation/test split to detect small improvements that matter in real-world use. Our work is the first attempt (that we know of) to develop methods to decrease the amount of data necessary to detect an improvement, rather than simply add more data.

With respect to the issue of detecting improvements in our models, much of the machine learning literature has tackled this problem. Early works explained that ordinary t-tests and other statistical methods are not reliable for machine learning cases for a variety of technical reasons [20]. More recent works have consistently found that a non-parametric Wilcoxon test is a reliable way to detect which algorithm performs best, if multiple trials (i.e., datasets) are available [21, 22, 23]. Other approaches to testing over the space of hyper-parameter values have also been proposed to better measure the improvement achieved, if any, by a new algorithm [24, 25]. The goal of our work is to provide a better foundation for using these prior model selection strategies, as simple cross-validation over an existing biased academic dataset is unlikely to produce a robust conclusion [26, 27].

2.1. Dataset

To perform our study, it was critical that we had a representative population of benign samples, as crawling publicly available sources has been demonstrated to produce models with insufficient

[1] This costs $400,000/year.

diversity, which do not generalize to new malware [9, 10, 2, 11]. Because our interest is in producing train/test splits that are also of a reasonable size, so that academics can use them, we use the EMBER 2018 dataset [19] which contains 300,000 training and 100,000 testing benign files. The EMBER dataset also includes malicious files, but they are not evenly distributed by malware family or type, which is problematic for our dataset construction approach.

For this reason, we use the VirusShare corpus [28] as a source of freely available malware. Malware family labels can also be obtained freely via the AVClass [29, 30] tool combined with the VirusTotal reports of [31]. Using these sources we are able to get hundreds of malware families with thousands of samples each. Following [32] we use the same top 184 most frequent malware families with 10,000 samples each, 8,000 for training and 2,000 for testing.

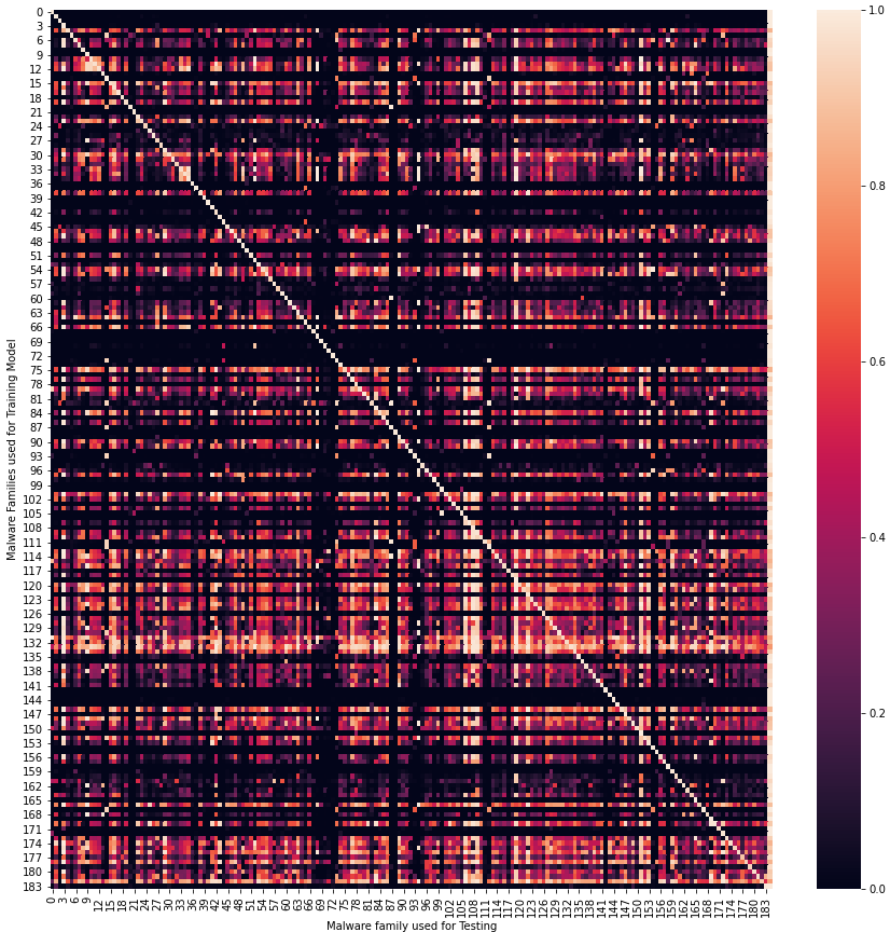

Figure 1: Here we show the cross-error rates of the MalConv models. Each row corresponds to the malware family used in training, and the columns show the recall rate (see color scale) against all the malware families. Dark horizontal bands show malware families that do not generalize to other malware, and vertical dark bands are malware families that are hard to generalize to. Conversely, white bands are easy to generalize from/to respectively. The XGBoost result is near-identical, showing strong correlations in malware family generalization behavior across model and feature types.

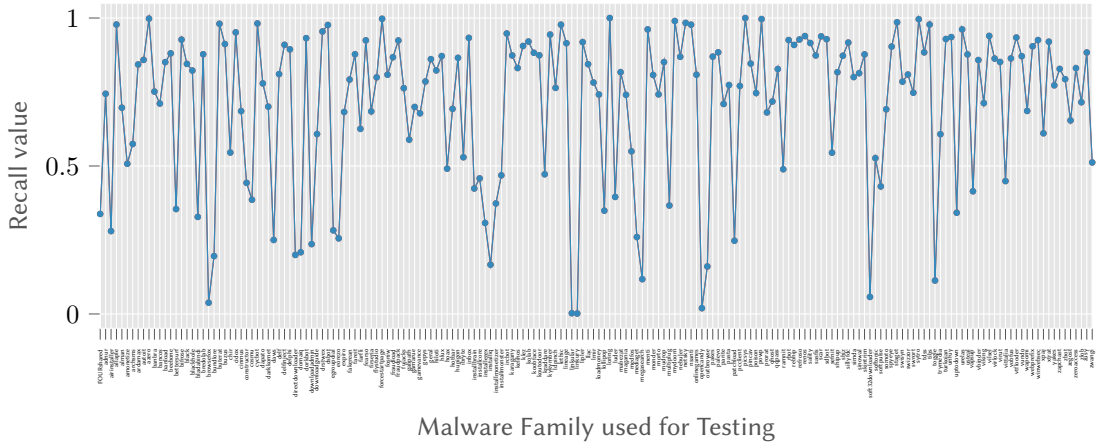

Figure 2: Example showing the malware recall rate (y-axis) of a model trained using the top five best malware families (in terms of highest average recall against other families) as reported by Figure 1. While the average recall rate is reasonably large, the recall per-family has an extremely high variance. This makes it challenging to determine if a performance difference comes from luck or true effect.

3. Approach

To make our benchmarks of configurable difficulty, we will start with the all-pairs cross-errors shown in Figure 1. Each row corresponds to selecting one of the 184 malware families to be the only family used during training. The resulting classifier is then tested on itself and all 183 of the other malware families, with the recall score in the corresponding columns. (The main diagonal shows the recall we get when testing on the same malware as was used for training). This gives us information on how useful each malware family is, on its own, in predicting all other malware families. In every case a random set of benign files is down-sampled to the same number of malicious files. That is, 8,000 malicious and 8,000 benign files are used to train a model for each row.

The goal will be to generate train/test splits into three categories mentioned earlier, namely Easy, Medium, and Hard. We will start by generating ten different train/test splits in each of the categories. Note here that each train/test split must have no overlap of malware families between the train and test splits, but different train/test splits might share some families. That is to say, the family "cycbot" may occur in training splits 1, 3, and 4, but that means the "cycbot" family cannot occur in test splits 1, 3, and 4. In this way every individual split is a meaningful test of generalization to new malware families, the ultimate goal of any malware detector. Each training split is trained on independently (not cross-validated), and so overlap between splits will not impact the results. So we will have 30 distinct train/test splits in total, ten each for the Easy, Medium, and Hard categories.

The algorithm we use to create these train/test splits is shown in Algorithm 1. The strategy is to apply a random search to obtain a set of training families T and a set of testing families V, which satisfy the constraint that none of the training families perform much better or worse than target recall threshold τ on any of the testing families. That is, $|M[t_i, v_j] - \tau| < \epsilon$ for $t_i \in T, v_j \in V$. This is done from the Malconv 184 x 184 data M by first identifying elements

Algorithm 1 Benchmark search

Require: 184×184 accuracy matrix M, target recall threshold τ, closeness parameter ϵ, max iterations I

1: $T, V \leftarrow \{\cdot\}, \{\cdot\}$ ▷ Training and validation sets
2: $C = \{(t_1, v_1), (t_2, v_2), \ldots\} \leftarrow \text{argwhere}(|M - \tau| \leq \epsilon)$
3: $i = 0$
4: **for** $i \in [1, \ldots, 10]$ **do**
5: Select a new (t_i, v_i) from C
6: **if** $t_i \in T$ or $v_i \in V$ **then**
7: Discard (t_i, v_i)
8: **if** $|M[t_j, v_i] - \tau| > \epsilon$ for any $t_j \in T$ **then**
9: Discard (t_i, v_i)
10: **if** $|M[t_i, v_j] - \tau| > \epsilon$ for any $v_j \in V$ **then**
11: Discard (t_i, v_i)
12: **if** (t_i, v_i) not discarded **then**
13: Add(T, t_i), Add(V, v_i)
14: **if** $i > I$ **then**
15: $\epsilon = \epsilon + 0.05$, then **go to** 2
16: **return** T, V

in the matrix which are ϵ-close to τ. The candidate pairs (t_i, v_i) of training and testing families corresponding to those elements are then randomly sampled.

At each iteration, while the sampled pair satisfies the performance constraint by design, it must also satisfy the constraint pairwise among all the families already selected in the training and testing sets. If this condition holds and neither member of the pair is already selected, then the pair is added to the growing training and testing sets. This procedure runs until 10 distinct families have been chosen for both training and testing. If the algorithm is unable to converge for ϵ, then ϵ is loosened (increased) and we try again. For efficiency, when this happens we do not discard the progress we have already made, and in line 2 we use the previous ϵ as a lower bound and the new ϵ as an upper bound for identifying new candidate pairs. For the Easy, Medium, and Hard splits we use $\tau = 0.9$, 0.5, and 0.25 respectively. We set the number of iterations $I = 1000$ and $\epsilon = 0.05$ throughout.

4. Results

In our tests, we use four models for evaluation. First is a byte 6-gram model that has been popular in academic malware detection research for several years [14, 9, 33]. Second we use MalConv [34] and its extended approach MalConvGCT [35]. Finally, we use the Ember feature vectors [19] with the XGBoost algorithm [36] as the standard domain knowledge approach, which we will refer to as just "XGBoost" for brevity. In our experiments, each train/test split we produce has 160,000 and 40,000 total samples, respectively. However, because within a

given train/test split no family is used for both training and testing, we note that if memory is a constraint, the experiments can be performed using just the training or testing sets alone. We remind the reader that randomly sampling to a split of this size will produce a model with $\geq 90\%$ accuracy in all cases.

Having defined our approach to generating harder train/test splits, we begin with the primary results as shown in Table 1. As can be seen, we are able to successfully produce datasets that are more challenging than the original dataset. With a lower baseline level of accuracy, it becomes possible to measure effect sizes with a moderate number of samples - and avoid the over 30 million files that are needed to reliably detect improvement of XGBoost like models on regular malware data [2].

Table 1
A set of Easy, Medium, and Hard train/test splits ("Modified" columns) created using Algorithm 1. The goal is to produce splits that have lower accuracy than the normal benign/malicious classification task ("Normal" column). In each case, we see that we successfully produce harder splits, which may allow detection of larger effects.

		Modified Train/Test		
Algorithm	Normal	Easy	Medium	Hard
Byte n-grams	94.87	79.48	66.06	58.52
MalConv	91.14	85.88	63.81	44.73
MalConv GCT	93.29	83.43	61.51	33.49
XGBoost	99.64	99.08	90.80	72.80

Recall that the same splits are being used for all four algorithms. This shows an unusual, but important, kind of generalization. Even though MalConv is less accurate in normal use than MalConvGCT, and significantly less accurate than domain-knowledge-wielding XGBoost, the Hard split is able to reduce XGBoost down to just 72.80% accuracy. This shows that our benchmark search is 1) finding correlations of intrinsic difficulty, and 2) allowing us to avoid overly biasing a test-set against a specific approach. That is to say, if we used an XGBoost model to produce the splits to evaluate an improved XGBoost, we may unfairly over-compensate by having produced a dataset split that is too difficult.

To show the consistency of our results in producing train/test splits of comparable difficult, we show the result for multiple splits as a function of how many epochs MalConvGCT has trained for in Figure 3. Here it is clear that each of the Easy, Medium, and Hard difficulty levels exhibit high degrees of similarity in their difficulty. This is important to avoid a naive solution where the target difficulty is obtained by averaging models that are *too* hard against others that are *too* easy. Such an undesirable scenario would make performing multiple trials to use statistical tests difficult, as the overly easy and hard splits would degrade to adding noisy samples[2] to the test and reduce the total power of the test to conclude if one method was really better than another [21].

[2]Because each model easily gets all the easy splits correct, and the hard splits all misclassified, making the differences between two models indistinguishable.

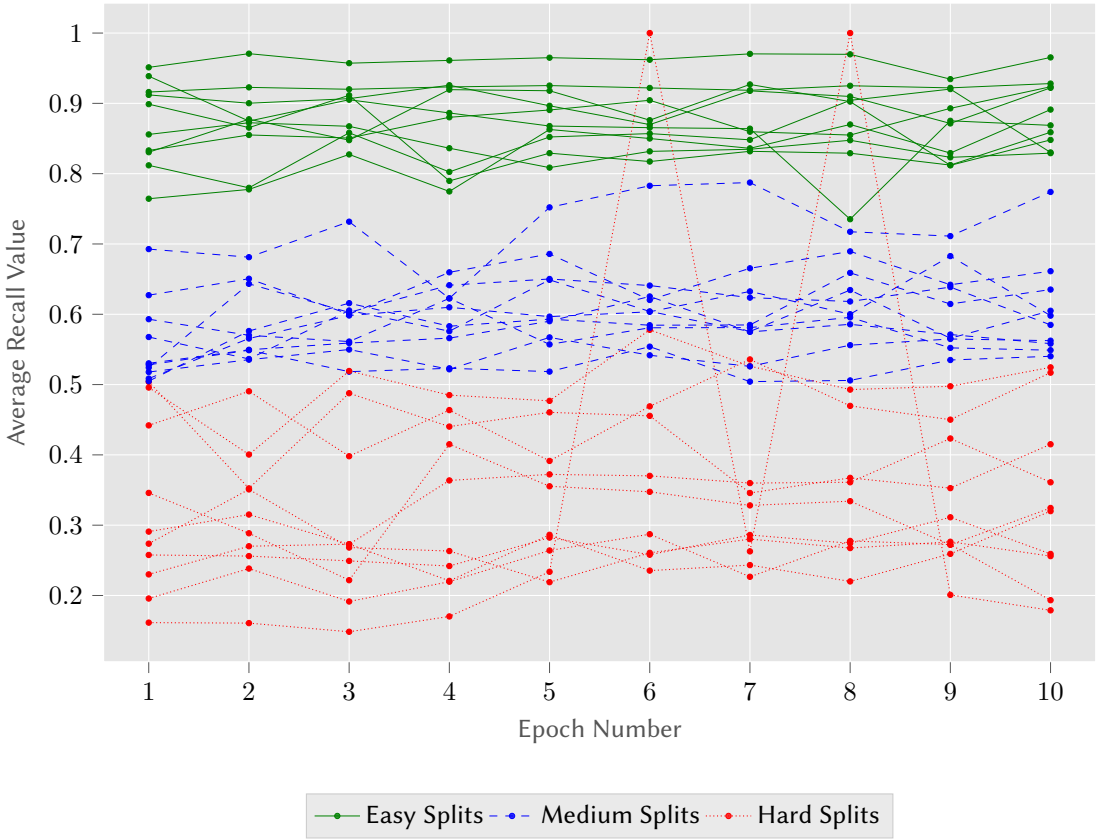

Figure 3: The malware Recall rate for the Easy (green), Medium (blue), and Hard (red) splits for MalConvGCT as training progresses. Note the two spikes are failure cases of the model marking all files as "malware" (100% false-positive rate). The results show how our approach produces multiple splits that are in a grouped range of similar difficulty.

4.1. Ablation

Having established the efficacy of our approach to producing datasets of the desired difficulty level, we will now demonstrate two alternative but intuitive strategies that do not meet our needs. In the case of a desired "Easy" benchmark, one may naively select the top-K "best" families from Figure 1, which have the highest average recall against other malware families. Second, one may similarly decide that a "Hard" dataset should be produced by selecting the families with the lowest average recall.

For the "Easy" case of selecting the top-K, we first show as an example the results of this strategy when picking the top-5 most frequent families in Figure 2. Though this produces a recall of 70%, the variance of the results is extremely high. This huge variance is undesirable for the same reason as our results from Figure 3. We want reasonably similar performance characteristics for each split to maximize the power of subsequent conclusions about improvement. Each overly easy and hard split is one that does not provide meaningful information to the question of whether a new algorithm would perform better.

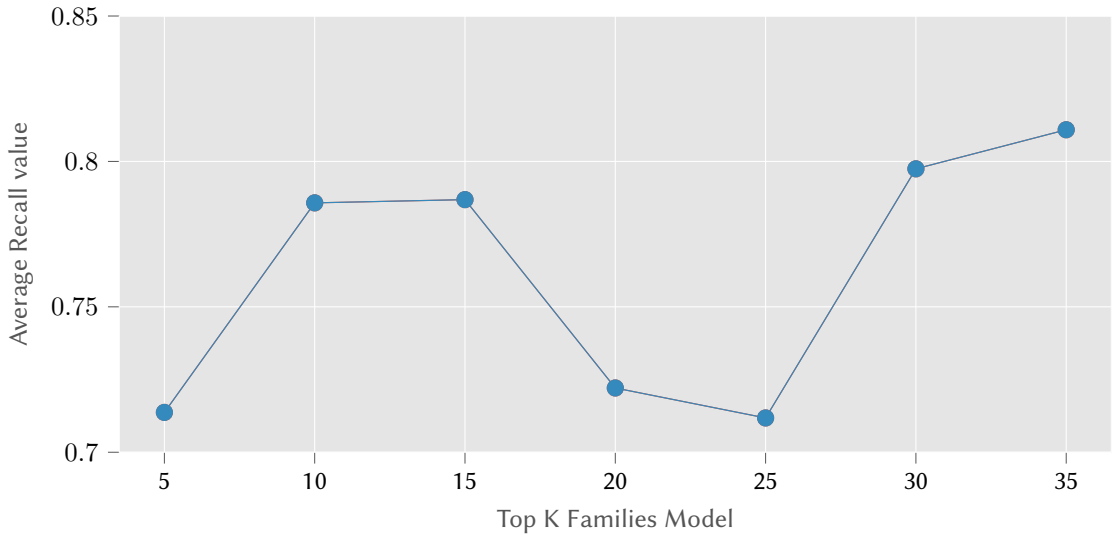

Figure 4: Average recall performance (y-axis) of MalConv Model trained on best k generalizing families (x-axis) across all 184 malware families. The performance of the model fluctuates between 71% and 81% in a non-monotonic fashion, making top-K selection unreliable to a specific level of performance, and with little total average variation. This fluctuation makes this strategy ineffective for building a benchmark of a desired level of difficulty.

One may wonder instead if the issue would improve by selecting more families. This is unfortunately not the case, and there is relatively little variation as the top-K is altered from $K = 5$ to $K = 35$, as shown in Figure 4 (We note that all values of K look qualitatively similar to Figure 2 as well).

A different kind of issue occurs when selecting the worst-K malware families to produce a "Hard" dataset. $K = 10$ is shown as an example in Figure 5, where the 10 chosen families each have 100% recall, and the model does not meaningfully learn to detect any of the remaining malware families. In this case, the hardest families are so distinct on their own that the model easily learns to overfit to the specific malware families, and the default for any other input becomes "benign". This is similar to the overly-strong data leakage signal discussed by [9] when building a benign dataset from scraping a clean install of Microsoft Word. We again note that using multiple values of K all result in qualitatively the same results for the worst-K strategy.

5. Conclusion

We have now shown it is possible to use malware family information to construct better train/test splits for benchmarking purposes, where the difficulty of the split is configurable. This was demonstrated with an Easy, Medium, and Hard split — and in all cases a weaker model is able to produce splits that are effective against a more powerful model. This is a necessary condition of utility, as the purpose of the splits is to test a hopefully more powerful alternative model. We further validate our approach by ablating against simpler design alternatives, which do not

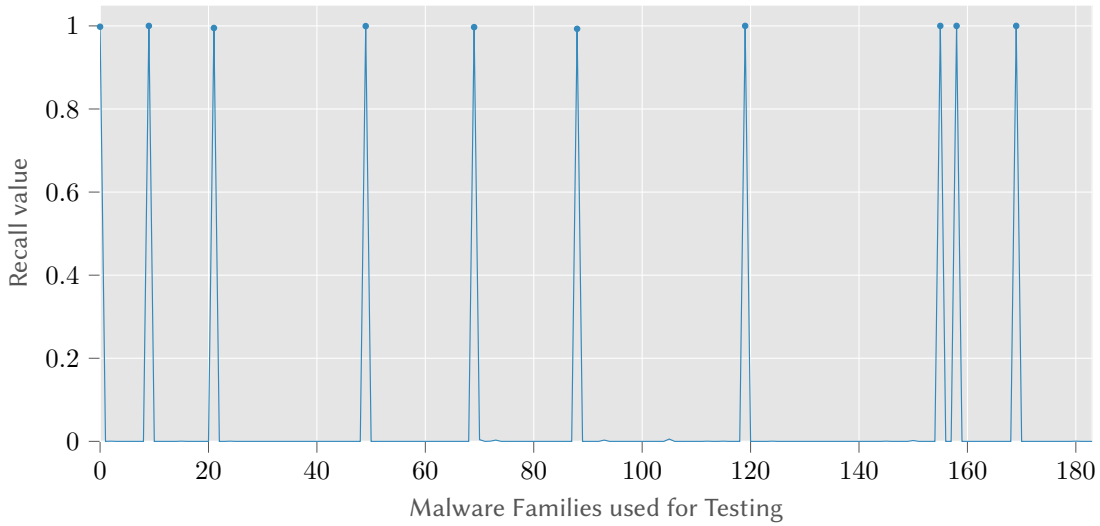

Figure 5: Malware family (x-axis, by ID number for space) and the recall of that family (y-axis) of a MalConv [34] Model trained on worst 10 families. The result has almost zero generalization to any other malware family, and the 10 high recalls of near 100% correspond to the 10 families used to train the model. This shows selecting the worst generalization families is too hard to construct a useful benchmark.

produce benchmarks of usable quality.

References

[1] J. F. Gantz, R. Lee, A. Florean, V. Lim, B. Sikdar, L. Madhaven, S. K. S. Lakshmi, M. Nagappan, The Link between Pirated Software and Cybersecurity Breaches How Malware in Pirated Software Is Costing the World Billions, Technical Report, IDC, 2014.

[2] R. Harang, E. M. Rudd, SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection, arXiv (2020). URL: http://arxiv.org/abs/2012.07634, arXiv: 2012.07634.

[3] A. T. Nguyen, E. Raff, C. Nicholas, J. Holt, Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints, in: IJCAI-21 1st International Workshop on Adaptive Cyber Defense, 2021. URL: http://arxiv.org/abs/2108.04081, arXiv: 2108.04081.

[4] K. Soska, C. Gates, K. A. Roundy, N. Christin, Automatic Application Identification from Billions of Files, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA, 2017, pp. 2021–2030. URL: https://doi.org/10.1145/3097983.3098196. doi:10.1145/3097983.3098196, series Title: KDD '17.

[5] G. E. Dahl, J. W. Stokes, L. Deng, D. Yu, Large-scale malware classification using random projections and neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 3422–3426. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6638293. doi:10.1109/ICASSP.2013.6638293.

[6] E. Raff, C. Nicholas, A Survey of Machine Learning Methods and Challenges for Windows Malware Classification, in: NeurIPS 2020 Workshop: ML Retrospectives, Surveys & Meta-Analyses (ML-RSA), 2020. URL: http://arxiv.org/abs/2006.09271, arXiv: 2006.09271.

[7] M. Eskandari, S. Hashemi, A graph mining approach for detecting unknown malwares, Journal of Visual Languages & Computing 23 (2012) 154–162. doi:10.1016/j.jvlc.2012.02.002.

[8] R. Perdisci, A. Lanzi, W. Lee, McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables, in: 2008 Annual Computer Security Applications Conference (ACSAC), IEEE, 2008, pp. 301–310. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4721567. doi:10.1109/ACSAC.2008.22.

[9] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, C. Nicholas, An investigation of byte n-gram features for malware classification, Journal of Computer Virology and Hacking Techniques (2016). URL: http://link.springer.com/10.1007/s11416-016-0283-1. doi:10.1007/s11416-016-0283-1.

[10] J. Seymour, How to build a malware classifier [that doesn't suck on real-world data], in: SecTor, Toronto, Ontario, 2016. URL: https://sector.ca/sessions/how-to-build-a-malware-classifier-that-doesnt-suck-on-real-world-data/.

[11] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, L. Cavallaro, TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time, in: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, Santa Clara, CA, 2019, pp. 729–746. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury.

[12] R. J. Joyce, D. Amlani, C. Nicholas, E. Raff, MOTIF: A Large Malware Reference Dataset with Ground Truth Family Labels, in: The AAAI-22 Workshop on Artificial Intelligence for Cyber Security (AICS), 2022. URL: https://github.com/boozallen/MOTIF. doi:10.48550/arXiv.2111.15031, arXiv: 2111.15031v1.

[13] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, S. R. White, Biologically Inspired Defenses Against Computer Viruses, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 985–996. URL: http://dl.acm.org/citation.cfm?id=1625855.1625983, series Title: IJCAI'95.

[14] J. Z. Kolter, M. A. Maloof, Learning to Detect and Classify Malicious Executables in the Wild, Journal of Machine Learning Research 7 (2006) 2721–2744. URL: http://dl.acm.org/citation.cfm?id=1248547.1248646, publisher: JMLR.org.

[15] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, S. Mukkamala, Malware Detection Using Assembly and API Call Sequences, J. Comput. Virol. 7 (2011) 107–119. URL: http://dx.doi.org/10.1007/s11416-010-0141-5. doi:10.1007/s11416-010-0141-5, publisher: Springer-Verlag New York, Inc. Place: Secaucus, NJ, USA.

[16] R. Zak, E. Raff, C. Nicholas, What can N-grams learn for malware detection?, in: 2017 12th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2017, pp. 109–118. URL: http://ieeexplore.ieee.org/document/8323963/. doi:10.1109/MALWARE.2017.8323963.

[17] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, T. Dumitraș, The Dropper Effect: Insights into Malware Distribution with Downloader Graph Analytics, in: Proceedings of the 22Nd

ACM SIGSAC Conference on Computer and Communications Security, ACM, New York, NY, USA, 2015, pp. 1118–1129. URL: http://doi.acm.org/10.1145/2810103.2813724. doi:10.1145/2810103.2813724, series Title: CCS '15.

[18] A. T. Nguyen, E. Raff, A. Sant-Miller, Would a File by Any Other Name Seem as Malicious?, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 1322–1331. URL: https://ieeexplore.ieee.org/document/9006132/. doi:10.1109/BigData47090.2019.9006132.

[19] H. S. Anderson, P. Roth, EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models, ArXiv e-prints (2018). URL: http://arxiv.org/abs/1804.04637, arXiv: 1804.04637.

[20] T. G. Dietterich, Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms, Neural Comput. 10 (1998) 1895–1923. URL: http://dx.doi.org/10.1162/089976698300017197. doi:10.1162/089976698300017197, publisher: MIT Press Place: Cambridge, MA, USA.

[21] A. Benavoli, G. Corani, F. Mangili, Should We Really Use Post-Hoc Tests Based on Mean-Ranks?, Journal of Machine Learning Research 17 (2016) 1–10. URL: http://jmlr.org/papers/v17/benavoli16a.html.

[22] J. Demšar, Statistical Comparisons of Classifiers over Multiple Data Sets, Journal of Machine Learning Research 7 (2006) 1–30. URL: http://dl.acm.org/citation.cfm?id=1248547.1248548, publisher: JMLR.org.

[23] R. Dror, G. Baumer, M. Bogomolov, R. Reichart, Replicability Analysis for Natural Language Processing: Testing Significance with Multiple Datasets, Transactions of the Association for Computational Linguistics 5 (2017) 471–486. URL: https://www.aclweb.org/anthology/Q17-1033. doi:10.1162/tacl_a_00074.

[24] X. Bouthillier, P. Delaunay, M. Bronzi, A. Trofimov, B. Nichyporuk, J. Szeto, N. Sepah, E. Raff, K. Madan, V. Voleti, S. E. Kahou, V. Michalski, D. Serdyuk, T. Arbel, C. Pal, G. Varoquaux, P. Vincent, Accounting for Variance in Machine Learning Benchmarks, in: Machine Learning and Systems (MLSys), 2021. URL: http://arxiv.org/abs/2103.03098, arXiv: 2103.03098.

[25] R. Dror, S. Shlomov, R. Reichart, Deep Dominance - How to Properly Compare Deep Neural Models, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 2773–2785. URL: https://aclanthology.org/P19-1266. doi:10.18653/v1/P19-1266.

[26] G. Varoquaux, Cross-validation failure: Small sample sizes lead to large error bars, NeuroImage 180 (2018) 68–77. URL: http://www.sciencedirect.com/science/article/pii/S1053811917305311. doi:https://doi.org/10.1016/j.neuroimage.2017.06.061.

[27] S. Varma, R. Simon, Bias in error estimation when using cross-validation for model selection, BMC Bioinformatics 7 (2006) 91. doi:10.1186/1471-2105-7-91, place: London.

[28] J.-M. Roberts, Virus Share, 2011. URL: https://virusshare.com/.

[29] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, AVclass: A Tool for Massive Malware Labeling, in: F. Monrose, M. Dacier, G. Blanc, J. Garcia-Alfaro (Eds.), Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Springer International Publishing, Paris, France, 2016, pp. 230–253. URL: http://dx.doi.org/10.1007/978-3-319-457

19-2_11. doi:10.1007/978-3-319-45719-2_11.

[30] S. Sebastián, J. Caballero, AVClass2: Massive Malware Tag Extraction from AV Labels, in: ACSAC, 2020. URL: http://arxiv.org/abs/2006.10615, arXiv: 2006.10615.

[31] J. Seymour, C. Nicholas, Labeling the VirusShare Corpus: Lessons Learned, in: BSidesLV, Las Vegas, NV, 2016.

[32] E. Raff, R. Zak, G. L. Munoz, W. Fleming, H. S. Anderson, B. Filar, C. Nicholas, J. Holt, Automatic Yara Rule Generation Using Biclustering, in: 13th ACM Workshop on Artificial Intelligence and Security (AISec'20), 2020. URL: http://arxiv.org/abs/2009.03779. doi:10.1145/3411508.3421372, arXiv: 2009.03779.

[33] E. Raff, W. Fleming, R. Zak, H. Anderson, B. Finlayson, C. K. Nicholas, M. Mclean, W. Fleming, C. K. Nicholas, R. Zak, M. Mclean, KiloGrams: Very Large N-Grams for Malware Classification, in: Proceedings of KDD 2019 Workshop on Learning and Mining for Cybersecurity (LEMINCS'19), 2019. URL: https://arxiv.org/abs/1908.00200.

[34] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, C. Nicholas, Malware Detection by Eating a Whole EXE, in: AAAI Workshop on Artificial Intelligence for Cyber Security, 2018. URL: http://arxiv.org/abs/1710.09435, arXiv: 1710.09435.

[35] E. Raff, W. Fleshman, R. Zak, H. S. Anderson, B. Filar, M. McLean, Classifying Sequences of Extreme Length with Constant Memory Applied to Malware Detection, in: The Thirty-Fifth AAAI Conference on Artificial Intelligence, 2021. URL: http://arxiv.org/abs/2012.09390, arXiv: 2012.09390 ISSN: 23318422.

[36] T. Chen, C. Guestrin, XGBoost: Reliable Large-scale Tree Boosting System, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. ArXiv: 1603.02754.