Title Page

NANYANG TECHNOLOGICAL UNIVERSITY

SCSE20-0460

DEEP LEARNING BASED MALWARE DETECTION USING HARDWARE PERFORMANCE COUNTERS

Submitted in Partial Fulfilment of the Requirements
for the Degree of Bachelor of Engineering (Computer Science)
of the Nanyang Technological University

by

Quah Yu Kiat

School of Computer Science and Engineering

2021

Contents Page

TABLE OF CONTENTS

Abstract

Studies in the past have investigated the feasibility of using HPCs (Hardware Performance Counters) as a metric to differentiate between benignware and malware. A major study titled “Hardware Performance Counters Can Detect Malware: Myth or Fact?” in 2018 concluded by using statistical models like Random Forest and Decision Tree that HPCs are not able to serve as a suitable metric. In the time since that study was published, newer deep learning models and techniques have been created. This paper first attempts to replicate the major study mentioned previously, then further investigate the feasibility of using HPCs as a metric with other models and techniques not used previously. LSTM (Long-Term Short Memory), Dense, and Ensemble models were investigated for their ability to use HPC values as a metric to differentiate between benignware and malware. This paper achieved results of ~80%, ~60%, and ~80% respectively for those models. Thus, this paper, based on the additional experiments done, supports the conclusion that HPCs are unable to reliably differentiate between benignware and malware. However, this paper provides the caveat that more data is needed for more experiments to be done to further support or contradict the conclusion that HPCs are an unsuitable metric. The source code used for this paper will also be made available to serve as an accessible base from which others can continue to build upon.

Acknowledgements

Ast/P Zhang Tianwei – Supervisor for this paper and provided guidance throughout this project

Ast/P Liu Weichen – Examiner for this paper

Boyou et al., 2018 – For data used in this paper and reference experiments within their paper

1. Introduction

1.1. Background

Despite major advancements in computer security in recent years, malware is still an ever-present threat to networks and personal devices alike. A key example was the SolarWinds hack[1] in 2020, where SolarWinds' Orion platform was compromised to distribute malware to users. Users of Orion includes, amongst others, FireEye – A US cybersecurity firm. The fact that even cybersecurity firms could be breached in such a manner emphasises the need for greater ability in detecting malware in systems before any damage can be done.

Currently, there are 3 major methods by which antivirus software detects malware – Signature checking, behavioural detection, and rule-based detection[2]. Signature checking involves checking against the antivirus firm's signature database (using hashing algorithms like MD5 and SHA-256) to find a match if any. Clearly, the first method does not protect against unknown malware which are not yet present in the firm's database. Behavioural and rule-based detection methods addresses that shortcoming by essentially detecting malware by their actions instead of their identity.

However, given the vast number of benign software currently available in the market, and the fact that benign programs are susceptible to being hijacked to perform malicious actions, the questions that presents are – "What are the fundamental differences between the actions of benign and malicious programs?", and "Can the fundamental differences, if any, be detected and thus clearly differentiating between benign and malicious programs without any knowledge with regards to the program's identity?". This paper addresses the second question.

[1] Constantin, L. (2020, December 15). SolarWinds attack explained: And why it was so hard to detect. Retrieved March, 2021, from https://www.csoonline.com/article/3601508/solarwinds-supply-chain-attack-explained-why-organizations-were-not-prepared.html

[2] Can antivirus detect malware? (2020, March 01). Retrieved March, 2021, from https://blog.reasonsecurity.com/2020/01/12/3316/

1.2. Objective

The next question would then be – "What metric should be used for comparison between programs benign or otherwise?". For the purposes of this paper, hardware performance counters will be the metric of interest. A major study titled "Hardware Performance Counters Can Detect Malware: Myth or Fact?"[3] (referred to as "original paper" throughout this paper) was published in 2018 and concluded that "micro-architectural level information obtained from HPCs [Hardware Performance Counters] cannot distinguish between benignware and malware". That study used simple statistical machine learning models (Decision Tree, Naive Bayes, Neural Net, AdaBoost, Random Forest, Nearest Neighbors) to attempt differentiating between benignware and malware.

Ever since that study was published in 2018, the field of machine learning has developed rapidly, and more sophisticated deep learning models and techniques have been created. Therefore, this paper aims to see if by using newly developed deep learning models and techniques, another conclusion different to the one in 2018 can be obtained.

[3] Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., & Joshi, A. (2018). Hardware Performance Counters Can Detect Malware: Myth or Fact? Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 457-468. doi:10.1145/3196494.3196515

2. Implementation

2.1. Data source

Due to the lack of hardware to perform independent data collection, data was obtained from the GitHub repository of the study "Hardware Performance Counters Can Detect Malware: Myth or Fact?" instead. As such, additional steps to verify data had to be taken and said steps will be detailed in section 3.2.

2.2. Platforms

Experiments were done using the free service platform Google Colaboratory[4] (Google Colab)

Code language – Python

Operating system – Windows 10

[4] Colaboratory - Google. (n.d.). Retrieved March, 2021, from https://research.google.com/colaboratory/faq.html

3. Work done

3.1. Data “collection”

The data available was in the form of 3 separate tarball parts, which had to be merged using the command ‘copy /B open_source_data.tar.gz.partaa + open_source_data.tar.gz.partab + open_source_data.tar.gz.partac open_source_data.tar.gz’. Note that the command differs from the given command in the repository’s readme file due to the different operating systems (Windows vs Linux).

The data in the resulting tarball was extracted using 7-zip. Within the source codes of the original study, there was usage of a dataset (amd_benign_metadata/benign_post_pca_data.txt) which was missing from the extracted data.

For easier access by Google Colab, the data extracted was uploaded to a personal Google Drive account which was mounted for usage.

3.2. Data verification

Before the data could be used in this paper’s experiments, it must first be verified that it is the same data used in the original paper’s experiments. By doing so, the original results could be reliably compared with this paper’s results. This was achieved by replicating the experiments done in the original paper with the parameters present in the original source codes.

Additionally, due to the absence of a dataset mentioned in original source codes, data verification is necessary to determine if the remaining data is representative of all datasets combined. Should similar results be obtained when replicating the experiments, it would demonstrably show that the missing dataset is not a major issue affecting experiment reliability.

3.3. New experiments

Once data verification is completed, and the original data is determined to be reliable for use in this paper's experiments, new models are built and tested on the original data.

A list of the models built are as follows:

- 1-3 LSTM layers with dropout = 0.2, 10-fold cross-validation, scaling applied
    - Optimizer used: Adam
    - Loss function used: Binary Cross Entropy
- 1-3 Dense layers with dropout = 0.2, 10-fold cross-validation, scaling applied
    - Optimizers used: Adam and SGD
    - Loss function used: Binary Cross Entropy
- Ensemble model (Max voting) using Random Forest, Decision Tree, AdaBoost classifiers
- Ensemble model (Blending) using Random Forest, Decision Tree, AdaBoost classifiers

The rationale behind choosing only Random Forest, Decision Tree, and AdaBoost classifiers for usage in the ensemble models will be explained in section 6.2.1.

3.4. Model structures

In summary, 4 model types will be used: Dense, LSTM, Ensemble (Max Voting), and Ensemble (Blending). The structure of these models will be displayed with a brief explanation on how it functions.

3.4.1. Dense

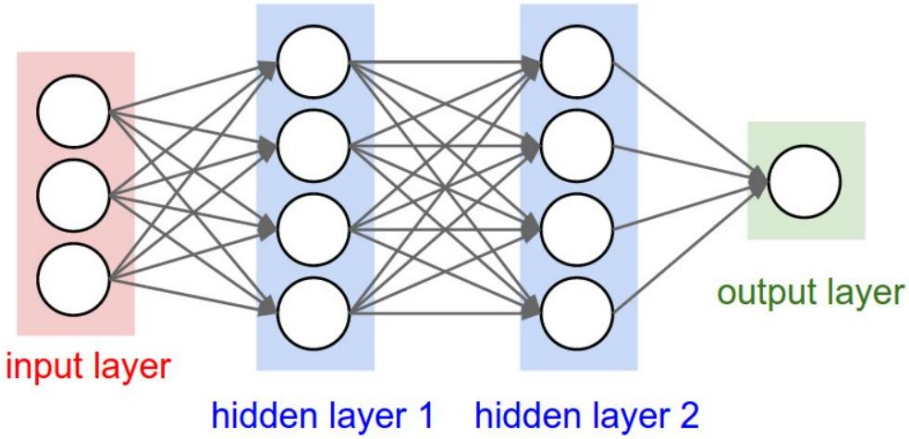


Figure 1: Neural network with 2 hidden Dense layers[5]

A Dense model essentially just a model using only Dense hidden layers. A Dense layer is a fully connected layer whereby each neuron (white circle in the figure) receives inputs from every neuron present in the previous layer, hence "fully-connected", as illustrated above. Each neuron receiving inputs then performs mathematical operations to the values it receives and outputs a result.

[5] [A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer.]. (n.d.). CS231n Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/neural-networks-1/

3.4.2. Ensemble

Like the name “ensemble” suggests, it involves combining the results of multiple models to produce a new result. This is illustrated in the following image:

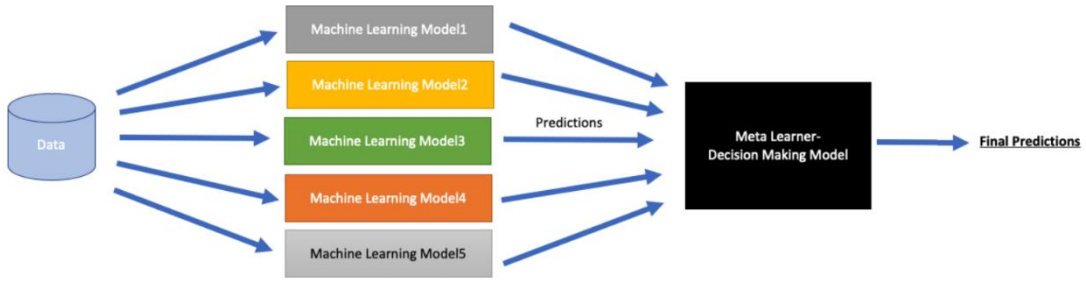


Figure 2: Ensemble models[6]

Machine Learning Models 1-5 shown in the figure above are also otherwise known as level-0 models, while the Decision Making Model is otherwise known as a level-1 model.

For Ensemble (Max Voting), the level-1 model is essentially just picking the most frequent prediction. For example, if out of 3 models, 2 models have the same prediction while the remaining has a different prediction. The former prediction will be chosen as the final prediction. Since only 3 component models were used, there would not be any case of a tie vote. Thus, the final result would be definitive.

For Ensemble (Blending), the level-1 model chosen was logistic regression for ease of usage and proof of concept. The level-0 component results were fed to a logistic regression model which then produces its own result.

[6] Gaurika Tyagi. (2020, August 17). Stacked ML Model [Illustration]. Ensemble Models for Classification. https://towardsdatascience.com/ensemble-models-for-classification-d443ebed7efe

3.4.3. LSTM

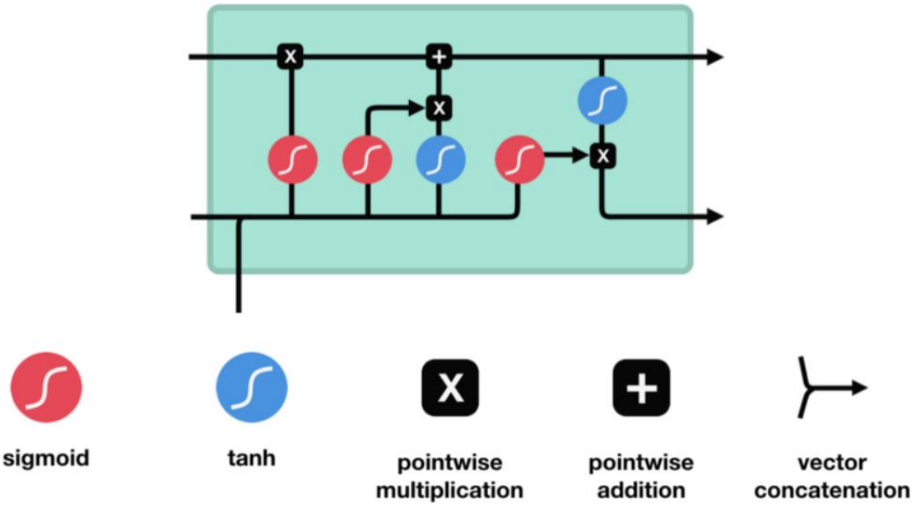


Figure 3: LSTM cell[7]

LSTM stands for long short-term memory. An LSTM network is an example of a Recurrent Neural Network (RNN) consisting of LSTM cells. Information is passed from one LSTM cell to the next until eventually a result is determined. An LSTM cell has 3 main components: An input gate, an output gate, and a forget gate. Each gate is comprised of a sigmoid function and a pointwise multiplication operation as shown in the figure above. The cell state is the upper horizontal line in the figure above that connects all cells and is altered depending on the gates in each cell.

[7] Michael Phi. (2018, September 25). LSTM Cell and It's Operations [Illustration]. Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

4. Comparisons with past work

Apart from the obvious difference in models used in the experiments, there are a few other key differences.

4.1. Setting a random seed

The original paper performed random shuffling of the dataset for each of the 1000 rounds of 10-fold cross validation they performed. The seed used was int(round(time.time())) which is dependent on the machine's time during the experiment itself. This is not reproducible. While the intention of random shuffling was to ensure differences across the 1000 rounds performed, it does not aid in future reproducibility. Additionally, the results obtained from fixing the seed should be representative of the result obtained even with 1000 rounds of random shuffling. This is confirmed in section 5.1.

4.2. Data scaling

A quick investigation into the data used reveals a massive range of values present with regards to the training parameters present.

- Largest value = 391104353201424.5 (3.91×10^{14})
- Smallest value = -2798749567463115.0 (-2.80×10^{-15})
- Standard deviation = 232179958528530.25 (2.32×10^{14})
- Mean = -75007212932411.5 (-7.50×10^{13})

Standard deviation measures, in simple terms, how values are spread out across the dataset. The mean represents the average of all values present in the dataset.

The excessively large values would result in unexpected behaviour when fed into deep learning models, hence scaling is performed. Despite the scaling, the data should still be representative of the original since all values are scaled down equally.

This is proven by calculating the variables shown above again after scaling is done. Note that all values were reduced by a factor of 10^{14}.

- Largest value = 3.911043532014245
- Smallest value = -27.98749567463115
- Standard deviation = 2.321799585285302
- Mean = -0.7500721293241152

As reflected, even the standard deviation and mean were reduced by a factor of 10^{14}, thus proving that scaling does not alter any pre-existing relationships present within the data. In results, this will be proven again via another method.

5. Results

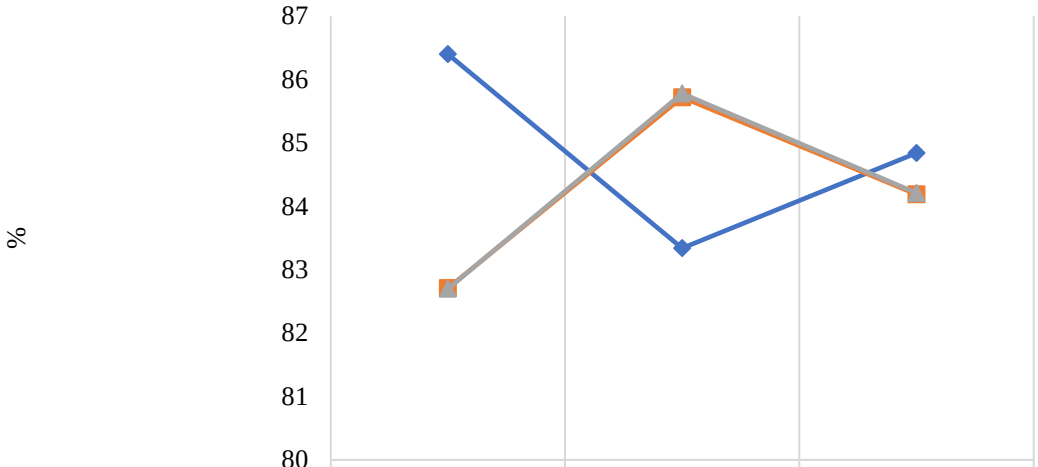
This paper's experiments mimic the original paper's TTA1 approach. The rationale behind choosing this approach is the idea that malware generally all behave in the same way. This is reflected in the Cyber Kill Chain framework[8] developed by Lockheed Martin. The 7 steps are: Reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives. Therefore, if malware generally act in the same manner, the HPC values produced during their operations should also theoretically, be comparable.

The original paper's results with the TTA1 approach is shown alongside this paper's results when attempting to replicate the original experiments.

[8] Cyber kill chain®. (n.d.). Retrieved March, 2021, from https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

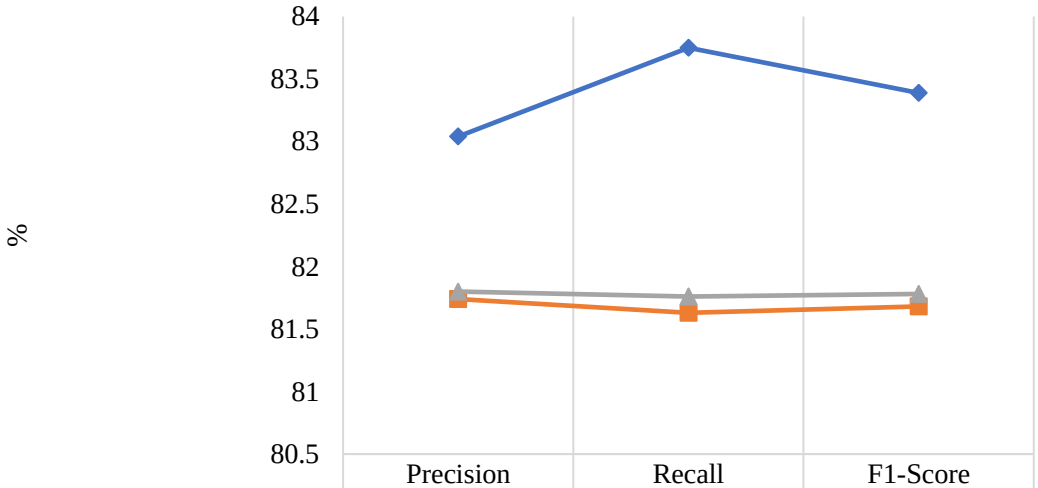
5.1. Data verification results

Classifier: Random Forest



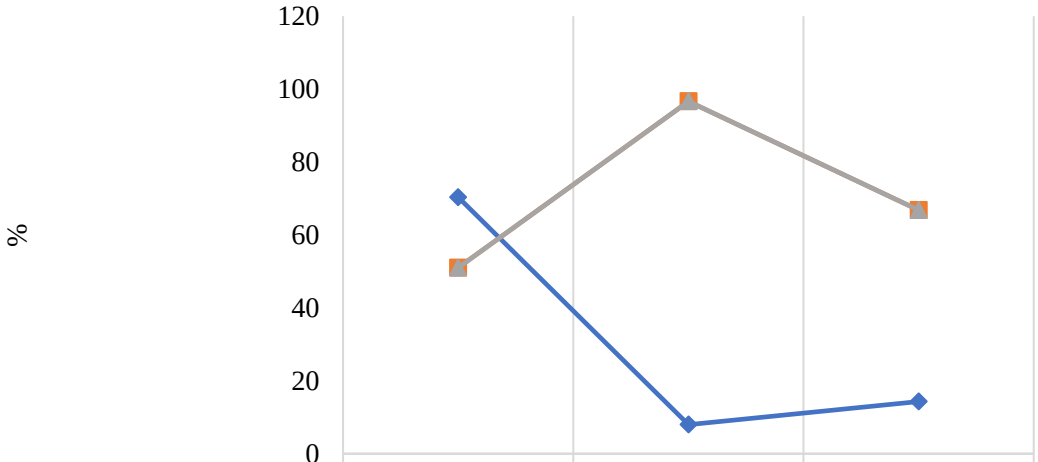
	Precision	Recall	F1-Score
Original paper	86.4	83.34	84.84
This paper (no scaling)	82.71	85.72	84.19
This paper (scaling)	82.7	85.78	84.21

Classifier: Decision Tree



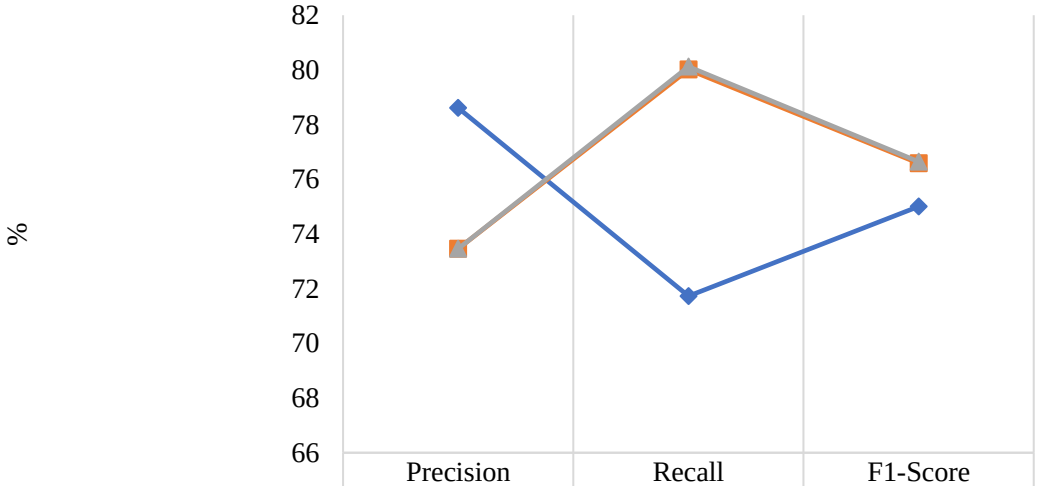
	Precision	Recall	F1-Score
Original paper	83.04	83.75	83.39
This paper (no scaling)	81.74	81.63	81.68
This paper (scaling)	81.8	81.76	81.78

Classifier: Naïve Bayes



	Precision	Recall	F1-Score
Original paper	70.36	7.97	14.32
This paper (no scaling)	51.03	96.62	66.79
This paper (scaling)	51.02	96.62	66.79

Classifier: AdaBoost



	Precision	Recall	F1-Score
Original paper	78.61	71.73	75.01
This paper (no scaling)	73.46	80.01	76.59
This paper (scaling)	73.46	80.12	76.64

Precision measures the proportion of positive class predictions that are correct, while recall measures the proportion of actual positives that were predicted correctly. The F1 score is essentially a weighted average of precision and recall. The way scikit-learn calculates these metrics are displayed in their online documentation[9] and detailed as follows:

- Precision = tp / (tp + fp)
- Recall = tp / (tp + fn)
- F1 score = 2 * (precision * recall) / (precision + recall)
- where tp = number of true positives, fp = number of false positives, fn = number of false negatives

The rationale behind choosing the 4 classifiers for replicating results is the ease of replication using existing libraries. Experiments were also run with and without scaling to prove that scaling has no negative impacts on the data. As shown, with the sole exception of Naïve Bayes, the results of this paper's experiments done without scaling are generally within 1-5% of the original paper's results.

5.2. New experiments' results

Detailed results for Dense and LSTM models can be found at this Github link: https[:]//github[.]com/ElementForte/SCSE20-0460

5.2.1. Dense model

Model built: 1-2 Dense layers with dropout = 0.2, 10-fold cross-validation, scaling applied

Result: ~60%

5.2.2. LSTM model

Model built: 1-3 LSTM layers with dropout = 0.2, 10-fold cross-validation, scaling applied

Result: ~80%

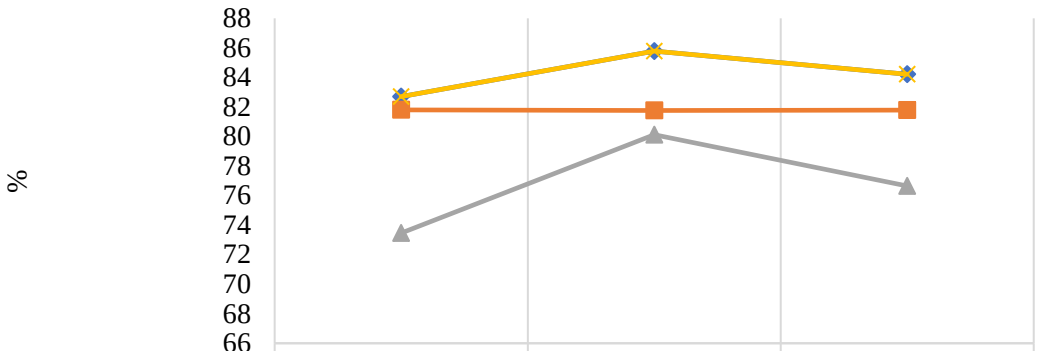
[9] API Reference. (n.d.). Retrieved March, 2021, from https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

5.2.3. Ensemble Model (Blending)

Level-0 models: Random Forest, Decision Tree, AdaBoost classifiers

Level-1 model: Logistic Regression

Results: ~80%, Little to no improvement compared to individual classifiers



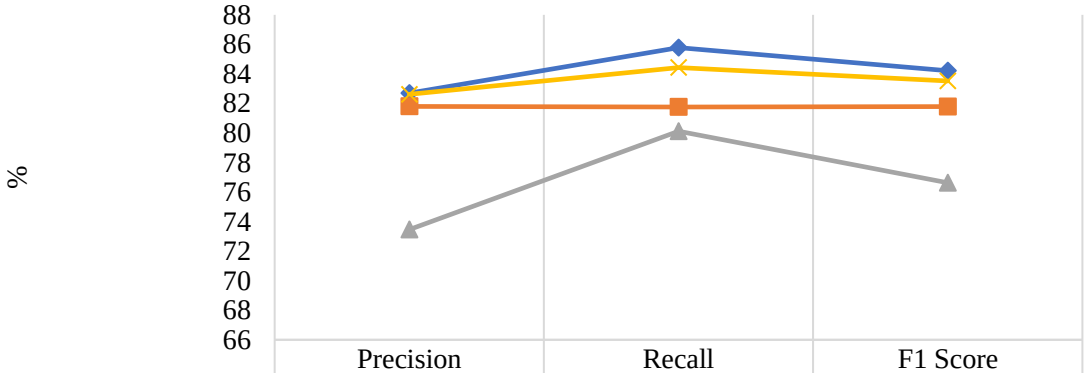
	Precision	Recall	F1 Score
Random Forest	82.7	85.78	84.21
Decision Tree	81.8	81.76	81.78
AdaBoost	73.46	80.12	76.64
Ensemble Model	82.7	85.78	84.21

5.2.4. Ensemble Model (Max Voting)

Level-0 models: Random Forest, Decision Tree, AdaBoost classifiers

Level-1 model: Logistic Regression

Results: ~80%, Little to no improvement compared to individual classifiers



	Precision	Recall	F1 Score
Random Forest	82.7	85.78	84.21
Decision Tree	81.8	81.76	81.78
AdaBoost	73.46	80.12	76.64
Ensemble Model	82.62	84.43	83.52

6. Discussion of results

6.1. Data verification

As mentioned in an earlier section, with the sole exception of Naïve Bayes, the results of this paper's experiments done are generally within 1-5% of the original paper's results. This was determined to be sufficiently similar to the original paper's experiments such that that data is sufficiently reliable to be used in this paper's experiments. However, there are 2 issues that must be addressed: The reason behind accepting a 1-5% absolute difference, and the reason behind a wildly different Naïve Bayes result.

6.1.1. 1-5% difference

For future replicability, a seed of 1 was set as the random seed for data shuffling and other relevant functions within the source code. This contrasts with the original paper's seed of int(round(time.time())), which is not reproducible. Despite this, the results from both papers running the same experiments should still be relatively comparable.

When ran multiple times with a fixed seed of 1, the results obtained were within 1-2% of each other. Additionally, the missing dataset could have contributed to a different result. Therefore, the absolute difference of 1-5% between the original paper's results and this paper's results was deemed to be acceptable.

6.1.2. Naïve Bayes difference

While it is unclear as to what caused the Naïve Bayes result to differ so wildly, a possible reason could be due to the previously mentioned missing dataset.

The "naïve" part of "Naïve Bayes" comes from the "assumption of conditional independence between every pair of features given the value of the class variable"[10]. It is possible that the missing dataset contains data that contributes to that assumption, and hence the absence of it resulted in a poor performance. However, this cannot be confirmed and is therefore just speculation.

As an aside, the missing dataset could have contributed to the difference in the previous section as well.

[10] 1.9. Naive Bayes. (n.d.). Retrieved March, 2021, from https://scikit-learn.org/stable/modules/naive_bayes.html

6.2. New experiments

6.2.1. Dense models

Dense models performed poorly at roughly 60%. This is most likely due to the type of data being fed into the model. As mentioned in section 6.2.2., the data available resembles a time series data set. Thus, Dense models performed poorly compared to LSTM models. Changing the optimizer or lowering the learning rate did not seem to produce any noticeable improvements. This result might be an indicator that future attempts at revisiting this topic would be better served by focusing on LSTM models or similar Recurrent Neural Network models.

6.2.2. Ensemble models

The rationale behind implementing ensemble models is that by combining the results of multiple base models, the result could be better than any of the individual base models. In fact, ensemble models "tend to strongly outperform their component models on new data" [11] . Naturally, the better the performance of the component models, the better the performance of the ensemble model. Thus, the simple statistical models were chosen as the component models for the ensemble model, as they performed equal to or better than the other models created in other experiments. Naïve Bayes model results were omitted due to relatively poor precision and F1 score.

The output of the component level-0 models would be used as input into the level-1 model, which in turn produces its own output. Thus, each row of data (each sample) would be represented in the level-1 model by three '1' or '0' values obtained from each component level-0 model.

Unfortunately for the parameters and methods chosen in this paper's experiments, the ensemble models were not able to perform any significantly better than the component level-0 models.

[11] Nisbet, R., Miner, G., Yale, K., Elder, J. F., & Peterson, A. F. (2018). Handbook of statistical analysis and data mining applications. London: Academic Press.

6.2.3. LSTM models

All LSTM models gave a result of roughly 80%. This held consistent even when variables such as batch size and number of LSTM layers were changed. Loss values were also relatively stable with changing batch sizes and number of layers. A result of approximately 80% makes LSTM models slightly worse than the statistical models used when replicating results of the original paper. However, this result has the potential to be improved.

LSTM models work well when provided with time series data, whereby data are listed according to the flow of time. In the context of HPC values, an ideal time series dataset would be HPC values sampled at regular time intervals from when the malware is first introduced, to an arbitrary point of choice which could be when the malware fulfils its original intended purpose (for example, data encryption for ransomware). Different run times of different programs could be addressed by normalisation of the data.

The way in which the original paper collected data resembles a time series. The total run time of each program was divided into 32 intervals, while 6 micro-architectural events were recorded. This provides possibilities in other ways data could be collected. The original paper's setup utilised the AMD Bulldozer micro-architecture which allowed them to monitor 130 micro-architectural events 6 at a time. Given that the hardware used was released in 2011, more advanced hardware that have been released since could allow for more events to be monitors at a time, thereby providing more data for input into machine learning models. The number of intervals could also be varied and tested to find an ideal value.

7. Conclusion

7.1. Gaps and limitations

This paper was not intended to be the be-all and end-all paper with regards to the models used and results obtained. Instead, it serves as a base from which others can continue building upon. Examples of limitations include the limited number of parameters used in building the models in this paper. Due to time constraints, only a small range of parameters could be experimented with.

Additionally, the data used in this paper was only from one source and was not collected independently. There are two flaws with this approach.

Firstly, since there were no other data made available open source, the results obtained in this paper was unable be tested on other datasets. Thus, the assumption had to be made that the data obtained was legitimate and accurate. Having minimally one other dataset from another source to cross validate results would greatly improve the confidence in any results obtained.

Secondly, and as an extension of the first flaw, by not collecting the data independently, not only does the assumption of accuracy and legitimacy must be made, raw data of the actual HPC values is also unavailable. It is not impossible that if the raw data were handled differently for newer models, a better performance could be obtained. An ensemble model that has component level-0 models taking in separate various HPC values could potentially produce a better result.

7.2. Future work

As the source code for this paper can be run in Google Colaboratory without any additional resources or minimum system requirements, it would be trivial for others to continue from where this paper has left off. The source code (and results from this paper) can be found at this Github link: https[:]//github[.]com/ElementForte/SCSE20-0460.

Future work can be grouped into two categories: Models and data.

7.2.1. Models

Similar to how this paper implemented LSTM and Ensemble models, while the original paper implemented simple statistical models like Random Forest and Decision Tree, newer models that are created in the future could be tested again. Other current models could also be used in new experiments to try for better results. Additionally, different parameters like varying batch sizes, number of neurons, and layer depth could be experimented with. There might also be the need to replicate this paper's experiments to compare the results to verify that the models in this paper's experiments were built properly.

7.2.2. Data

There is a need for a new set of data to be collected, either by replicating the original paper's experiments, or via an entirely new methodology. Having multiple sets of data producing similar results would greatly improve the reliability of the results, and should vastly differing results be obtained, being able to pinpoint which methods are preferred would help in future studies of this topic. Currently, based on the experiments done thus far, HPCs indeed seem unable to differentiate accurately and reliably between benignware and malware. However, with more datasets and methods, that conclusion can either be further determined, or be disproved altogether.

References

[1] Constantin, L. (2020, December 15). SolarWinds attack explained: And why it was so hard to detect. Retrieved March, 2021, from https://www.csoonline.com/article/3601508/solarwinds-supply-chain-attack-explained-why-organizations-were-not-prepared.html

[2] Can antivirus detect malware? (2020, March 01). Retrieved March, 2021, from https://blog.reasonsecurity.com/2020/01/12/3316/

[3] Zhou, B., Gupta, A., Jahanshahi, R., Egele, M., & Joshi, A. (2018). Hardware Performance Counters Can Detect Malware: Myth or Fact? Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 457-468. doi:10.1145/3196494.3196515

[4] Colaboratory - Google. (n.d.). Retrieved March, 2021, from https://research.google.com/colaboratory/faq.html

[5] [A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer.]. (n.d.). CS231n Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/neural-networks-1/

[6] Gaurika Tyagi. (2020, August 17). Stacked ML Model [Illustration]. Ensemble Models for Classification. https://towardsdatascience.com/ensemble-models-for-classification-d443ebed7efe

[7] Michael Phi. (2018, September 25). LSTM Cell and It's Operations [Illustration]. Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[8] Cyber kill chain®. (n.d.). Retrieved March, 2021, from https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[9] API Reference. (n.d.). Retrieved March, 2021, from https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

[10] 1.9. Naive Bayes. (n.d.). Retrieved March, 2021, from https://scikit-learn.org/stable/modules/naive_bayes.html

[11] Nisbet, R., Miner, G., Yale, K., Elder, J. F., & Peterson, A. F. (2018). Handbook of statistical analysis and data mining applications. London: Academic Press.