Navigating Cyber Threats: A Deep Dive into ML Model Performance

Alex Vazquez
University of British Columbia
Dept. of Electrical and Computer Engineering

Abstract

This project builds upon the findings discussed in my previous paper, "Cybersecurity in the Machine Learning Lens: Challenges and Solutions." The goal of this research is to empirically validate and expand upon the machine learning techniques explored in that paper, specifically in the context of cybersecurity. To achieve this, I collected data through custom-built web crawlers, the VirusTotal API, and existing malware datasets. After thorough data cleaning and preprocessing, I applied a variety of machine learning models, including Random Forest Classifier, Linear SVC, Logistic Regression, and Convolutional Neural Networks (CNN) with different optimizers. The performance of each model was evaluated to determine the most efficient approach, thereby testing and verifying the claims made in my previous research.

1 Data Collection

After reviewing the paper by Amit, Idan [2], I requested access to the dataset referenced in the study. Although I received confirmation of my request, the dataset was never provided. The paper emphasizes the scarcity of publicly available malware datasets and calls on private companies and organizations to share anonymized data to benefit researchers and the public. Motivated by this challenge, I decided to take the initiative and develop my own dataset.

1.1 Dataset building

The development of the Malware4Alex dataset was a comprehensive process that involved multiple stages of data collection, processing, and refinement. This process is visually summarized in Figure 1, which outlines the steps taken to create a dataset tailored specifically for malware analysis.

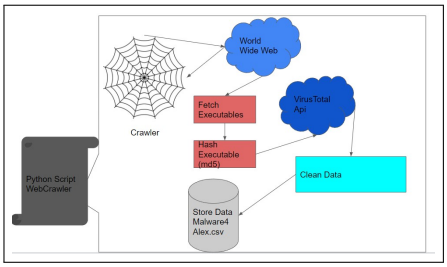


Figure 1: Diagram of Data Collection process for Malware4Alex

The creation of the Malware4Alex dataset began with the deployment of a Python script on an AWS EC2 container. The choice of AWS EC2 was driven by its scalability and reliability, allowing the script to run continuously in the background with minimal intervention. The script was designed to systematically crawl the web for executable files, both benign and malicious, which are crucial for building a comprehensive malware dataset. By targeting a wide array of sources, the script aimed to capture diverse samples that represent the different forms of malware encountered in the wild.

Once these executables were identified and downloaded, they were transferred into a controlled environment using VirtualBox. This virtualized environment was essential for ensuring that the malware samples could be safely analyzed without posing any risk to the underlying system or network. Within this environment, each executable was hashed using cryptographic algorithms to create a unique identifier. This hashing process not only facilitated the efficient management and indexing of the files but also ensured the integrity and uniqueness of each sample.

Following the hashing process, the script utilized the VirusTotal API to gather detailed information about each executable. VirusTotal is a widely recognized platform that aggregates data from various antivirus engines and other tools to provide a comprehensive analysis of files. By leveraging this API, the script was able to retrieve critical metadata, including the detection ratio, behavioral analysis, and potential indicators of compromise associated with each executable. This information is invaluable for training machine learning models, as it provides rich contextual data that enhances the accuracy and effectiveness of the models.

After collecting this data, the script entered the data cleaning and preprocessing phase. During this stage, the raw data was carefully examined to identify and remove incomplete or corrupted entries. Data cleaning is a vital step in the dataset creation process, as it ensures that only high-quality, relevant data is included in the final dataset. Preprocessing involved normalizing the data, converting categorical variables into numerical formats, and filtering out any noise that could negatively impact the performance of the machine learning models.

The cleaned and preprocessed data was then organized and saved in a CSV file format, making it easily accessible for further analysis and model training. Given the complexity and volume of the data, this structured format facilitated efficient data handling and ensured compatibility with various machine learning frameworks.

Despite the robust data collection and processing mechanisms, the project faced limitations in computing power and time, which necessitated the use of an existing dataset for the immediate analysis. However, the script continues to operate, progressively building the Malware4Alex dataset. As more data is collected, the dataset will grow in size and diversity, providing an increasingly rich resource for future research and model development.

The Malware4Alex dataset will be made available to the public in my Github repository once it reaches a substantial size. By sharing this dataset, the project aims to contribute to the broader cybersecurity community, enabling other researchers to benefit from the collected data and furthering advancements in malware detection and analysis.

1.2 PacktPublishing Dataset

Due to time and computing constraints, we opted to use the MalwareData.csv dataset provided by Packt Publishing for this project. This dataset was chosen for its convenience, as it comes pre-processed and includes 56 features along with 128,047 data points. The features in the dataset encapsulate various attributes of the malware samples, such as their operational behavior, system calls, and file properties, which are crucial for machine learning-based malware detection.

To prepare the data for model training, we conducted an initial cleaning process where we removed specific labels, such as "Name" and "MD5." These labels, while useful for uniquely identifying each sample, do not contribute to the predictive capabilities of the models. Instead, they act as identifiers and are unnecessary for our analysis beyond indexing the data. The dataset was then split into training and testing subsets to ensure proper model evaluation. The use of a pre-processed dataset allowed us to focus on model development and tuning, while still working with a robust and comprehensive dataset.

Afterward, we split the dataset into two subsets: (X_train, y_train) and (X_train, y_train) using the $(train_test_split())$ function from the scikit-learn library. The function was called as follows:

$$train_test_split(MalwareDataset, y, test_size = 0.2, random_state = 42)$$

This function split the data, allocating 20% of the samples for testing and 80% for training. This 80/20 split is a common industry standard, as it provides the model with a substantial amount of training data, which typically leads to better accuracy and generalization. By setting the random_state=42, we ensured the split was reproducible, allowing for consistent evaluation of model performance across different runs. The larger training subset enabled us to build models that could effectively learn patterns from the malware data, while the smaller test subset provided an unbiased evaluation of the models' performance on unseen data. We allocated 20% of the data for testing and 80% for training, a common industry standard since providing more training data typically improves model accuracy.

2 Model Building

In the realm of malware detection, a variety of machine learning techniques have been proposed and tested in previous studies. In particular, the approaches outlined in the paper by Mathew, Alex [1], and further elaborated in my reading report, highlight the effectiveness of both shallow and deep learning models for this task. The key techniques explored in these works can be categorized into two main groups:

1. Supervised Shallow Learning: This includes traditional machine learning models such as Random Forest, Support Vector Machines (SVM), and Logistic Regression. These models are known for their efficiency and simplicity, often performing well on structured data with clearly defined features.
2. Supervised Deep Learning: This group comprises more advanced techniques, such as Convolutional Neural Networks (CNN), Fully Connected Neural Networks (FNN), and Recurrent Neural Networks (RNN). These models excel at capturing complex patterns in high-dimensional data, making them particularly suited for tasks like malware detection, where feature representation plays a critical role.

For this project, I focused on implementing the machine learning methods that were both discussed in class and highlighted in the literature. The goal was to compare their performance on the dataset used and assess whether the findings from the previous research hold true in this context.

2.1 Supervised Shallow Learning Models

To establish a performance baseline, we initially implemented three commonly used supervised learning models: RandomForestClassifier, LinearSVC, and LogisticRegression. These models were trained with default hyper-parameters to observe their out-of-the-box performance. Specifically, for the Random Forest model, the max_depth parameter was left at its default value, allowing nodes to expand until all leaves were pure or contained fewer than the minimum samples required for splitting. This approach provided an initial benchmark for comparison before introducing any model-specific optimizations.

The resulting performance metrics for each model are as follows:

1. Random Forest
   Accuracy score on train data: 99.99%
   Accuracy score on test data: 99.52%
   F1 score: 0.9922
2. LinearSVC
   Accuracy score on train data: 29.84%
   Accuracy score on test data: 30.27%
   F1 score: 0.46
3. LogisticRegression
   Accuracy score on train data: 70.15%

Accuracy score on test data: 69.72%
F1 score: 0%

The results indicate a clear disparity in performance among the models. Random Forest significantly outperformed the other models, achieving near-perfect accuracy and an excellent F1 score, which suggests strong generalization and balanced classification capabilities. On the other hand, LinearSVC struggled with accuracy and exhibited a low F1 score, indicating challenges in correctly classifying the malware data. Logistic Regression performed moderately well in terms of accuracy but failed to achieve a meaningful F1 score, likely due to its inability to handle imbalanced classes effectively. These initial results set the stage for further optimization and analysis in subsequent sections.

2.2 Supervised Shallow Learning hyper-parameter Tunning

After evaluating the initial performance of the models using default parameters, it became evident that further optimization was necessary, particularly for models like LinearSVC and Logistic Regression, which underperformed in the initial tests. To enhance the performance of these models, we employed hyper-parameter tuning using cross-validation and grid search.

Cross-validation was utilized to assess the robustness of the models by partitioning the training data into multiple folds, ensuring that the models were tested on various subsets of the data. This approach helped mitigate the risk of overfitting by providing a more comprehensive evaluation of each model's generalization capabilities. By averaging the performance across these folds, we obtained a more reliable estimate of each model's effectiveness.

Grid search was then applied to systematically explore a range of hyper-parameter values. For Random Forest, this included varying the number of trees ("n_estimators"), the maximum depth of the trees ("max_depth"), and the minimum samples required to split a node ("min_samples_split"). For LinearSVC, we focused on optimizing the regularization parameter ("C") and the kernel function, while for Logistic Regression, we adjusted the regularization strength and experimented with different solver algorithms.

The improved metrics after hyper-parameter tuning are as follows:

1. Random Forest
   Accuracy score on train data: 99.99%
   Accuracy score on test data: 99.52%
   F1 score: 0.9922
2. LinearSVC
   Accuracy score on train data: 29.84%
   Accuracy score on test data: 30.27%
   F1 score: 0.46
3. LogisticRegression
   Accuracy score on train data: 70.15%
   Accuracy score on test data: 69.72%
   F1 score: 0%

Despite the comprehensive search for optimal hyper-parameters, the performance improvements were minimal, particularly for LinearSVC and Logistic Regression. For Random Forest, the model's accuracy and F1 score remained at their high levels, indicating that the default parameters were already well-suited to the dataset. The minimal change after tuning suggests that Random Forest's robustness and its ensemble nature allow it to perform well without requiring extensive parameter adjustments.

In contrast, LinearSVC and Logistic Regression continued to underperform, even after tuning. The lack of significant improvement in accuracy and F1 scores suggests that these models might be inherently less capable of handling the complexities of the malware dataset, which likely involves imbalanced classes and non-linear decision boundaries. LinearSVC, in particular, struggled with the high-dimensional feature space, and the adjustments made through grid search did not sufficiently address these challenges.

The results highlight the importance of model selection and the limitations of certain algorithms when applied to complex, real-world data. While hyper-parameter tuning is a critical step in optimizing model performance, it is not always sufficient to overcome the inherent challenges posed by the dataset. These findings underscore the need for more sophisticated models or alternative approaches, such as feature engineering or data balancing techniques, to achieve better classification outcomes.

2.3 Supervised Deep Learning CNN Model

Next, I implemented a convolutional neural network (CNN) designed to effectively classify malware data as depicted in Figure 2. CNNs are particularly powerful for pattern recognition tasks due to their ability to automatically learn spatial hierarchies of features through convolutional layers. The architecture of this CNN consists of four sequential dense layers: three utilizing the ReLU activation function, which introduces non-linearity into the model, and one using Sigmoid for binary classification output. This combination of layers was inspired by pre-trained models commonly used for malware analysis.

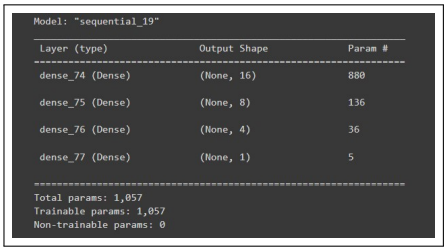


Figure 2: CNN Model Preview

The primary advantage of CNNs in malware detection lies in their ability to automatically learn relevant features from the data, eliminating the need for manual feature extraction. This makes them particularly suited for complex, high-dimensional data like executables. In this implementation, the CNN was trained using the Adam optimizer, which adapts the learning rate during training to improve convergence. After running the model for 10 epochs it yielded the following results:

CNN on Adam optimizer

Accuracy score on train data: 0.69%

While the initial performance was modest, this highlights the need for more careful tuning of the network's hyper-parameters and architecture to better adapt to the specific characteristics of the malware dataset. A key strength of CNNs is their flexibility and ability to scale with the complexity of the task, but this requires proper configuration.

2.4 Supervised Deep Learning Hyper-parameter tunning

Tuning a CNN model is more complex than tuning shallow learning models due to the higher number of hyper-parameters involved. These include layer configurations, learning rates, and optimization techniques, all of which interact in ways that can significantly impact the model's performance. Unlike traditional models that have relatively few hyper-parameters, CNNs require tuning across various dimensions, such as the number of convolutional layers, kernel sizes, activation functions, and regularization techniques like dropout. Each of these factors can dramatically influence the model's ability to generalize from the training data, making hyper-parameter tuning a critical step in achieving optimal performance.

In this project, I tested the CNN model with different optimizers, aiming to find the configuration that maximized performance. After experimenting with various options, I found that "rmsprop" significantly improved the model's accuracy, raising it to 94.72% on the training data. Following additional data cleaning and refinement, the model's performance further improved, reaching 95.17% on the training set and 95.38% on the test set. These improvements underscore the importance of careful tuning and preprocessing in deep learning models to ensure they can handle the complexity of malware data effectively.

3 Conclusion

The comparison between the various machine learning models, both in their default configurations and after hyper-parameter tuning, demonstrates the critical role of model optimization in achieving high performance. As summarized in Figure 3, the Random Forest model emerged as the best-performing algorithm, maintaining an accuracy of 99.99% on the training data and 99.52% on the test data, with an F1 score of 0.9922. This consistent performance highlights Random Forest's robustness and ability to generalize well on the dataset used.

Models	Train Data Accuracy	Test Data Accuracy	F1-Score
Random Forest Classifier	• Not-Optimized:0.9999% • Optimized: 0.9999%	• Not-Optimized:0.9952% • Optimized: 0.9961%	• Not-Optimized:0.9922% • Optimized:0.99%
Convoluted Neural Network	• Adam:0.6987% • Rmsprop: 0.9536%	• Adam:0.6967% • Rmsprop: 0.9472%	• Adam:0.67% • Rmsprop: 0.91%
Logistic Regression	• Not-Optimized:0.7015% • Optimized: 0.7634%	• Not-Optimized:0.6972% • Optimized: 0.7012%	• Not-Optimized:0% • Optimized: 0%
LinearSVC	• Not-Optimized:0.2984% • Optimized: 0.30%	• Not-Optimized:0.3027% • Optimized: 0.3029%	• Not-Optimized:0.46% • Optimized: 0.32%

Figure 3: Results

In contrast, LinearSVC significantly underperformed, achieving only 29.84% accuracy on the training set and 30.27% accuracy on the test set, even after hyper-parameter tuning. The low F1 score of 0.46 further emphasizes its unsuitability for this particular problem. Logistic Regression performed moderately, with a training accuracy of 70.15% and a test accuracy of 69.72%. However, its F1 score of 0.00 suggests that it failed to classify some classes correctly, indicating a possible issue with class imbalance or model configuration.

In the deep learning domain, the Convolutional Neural Network (CNN) initially produced suboptimal results with an accuracy of just 0.69% on the training data when using the Adam optimizer. However, after exploring various optimizers, "rmsprop" yielded significantly better performance, with an accuracy of 94.72% on the training set and 95.38% on the test set. Further improvements were achieved through additional data cleaning and refinement, demonstrating the potential of CNNs when properly optimized and trained with clean data.

These results underline the importance of hyper-parameter tuning, data preprocessing, and model selection in machine learning. While Random Forest demonstrated superior performance in this study, the CNN model's improvements after tuning indicate that with further refinement, deep learning models may outperform traditional machine learning methods in certain contexts, such as when working with large and complex datasets.

3.1 Future Considerations

Going forward, it would be beneficial to explore advanced ensemble methods, such as Gradient Boosting Machines (GBMs) or XGBoost, which may offer improvements over the standard Random Forest model. Additionally, incorporating techniques such as oversampling or synthetic data generation could address potential class imbalances in the dataset, particularly for Logistic Regression.

For the deep learning models, conducting a more extensive hyper-parameter search and experimenting with different architectures, such as deeper CNNs or hybrid models combining recurrent and convolutional layers, could further enhance performance. Implementing transfer learning from pre-trained models on similar tasks could also reduce training time and improve generalization.

Finally, as the Malware4Alex dataset continues to grow, it will provide a richer source of data for future experiments. A comparative study using both the current dataset and the newly developed Malware4Alex dataset could offer insights into how dataset characteristics influence model performance. Further research into explainable AI techniques could also help demystify how these models make decisions, making them more trustworthy and actionable in real-world cybersecurity applications.

References

[1] Mathew, Alex Machine Learning in Cyber-Security Threats (January 19, 2021). ICICNIS 2020, Available at SSRN: https://ssrn.com/abstract=3769194 or http://dx.doi.org/10.2139/ssrn.3769194

[2] Amit, Idan & John Matherly & William Hewlett & Zhi Xu & Yinnon Meshi & Yigal Weinberger (2019) Machine Learning in Cyber-Security - Problems, Challenges and Data Sets.AAAI-19 Workshop on Engineering Dependable and Secure Machine Learning Systems, 2019. 1812.07858v3

[3] https://github.com/Alexvz2/MachineLearningMalware