Exploring the Efficacy of Machine Learning and Deep Learning Techniques for Malware Detection

Abhiroop Sarkar, Prateek Dutta, Saurabh Barse, Adnan Quraishee, Achamma Thomas
G H Raisoni College of Engineering, Nagpur, M.H, India

Abstract

Malware, or malicious software, is a significant cybersecurity threat that can cause severe damage to individuals, organizations, and society as a whole. As malware continues to evolve and become more sophisticated, traditional signature-based detection methods are becoming less effective. Machine learning, a subfield of artificial intelligence, has shown great promise in improving malware detection accuracy and addressing the challenges posed by advanced malware. In this research paper, we provide a comprehensive review of the existing literature on malware detection using machine learning techniques. We discuss various types of machine learning algorithms, feature extraction techniques, and evaluation metrics used in malware detection research. We also highlight the challenges and limitations of using machine learning for malware detection, including issues related to data imbalance, feature selection, and adversarial attacks. Finally, we identify potential future directions for research in this field, including the use of deep learning, ensemble methods, for improving the accuracy, robustness, and interpretability of malware detection models.

Keywords:- EMBER Dataset;Data pre-processing, Data Modeling, XGBoost, Random Forest Regressor;

Introduction

The rapid increase in the use of computer systems and the internet has led to a rise in the number of malware attacks. Malware refers to malicious software that is designed to harm computer systems or steal sensitive information from them. Malware can be spread through various means, such as email attachments, downloads from the internet, or through vulnerabilities in software. In recent years, machine learning has shown great promise in various fields, including malware detection. Machine learning models can learn the features of malware samples and classify them into different categories.

Malware is a type of software that is designed to harm computer systems or steal sensitive information from them. The use of malware has become widespread in recent years, and it poses a serious threat to the security of computer systems. Malware can be spread through various means, such as email attachments, downloads from the internet, or through vulnerabilities in software. To prevent malware from infecting computer systems, there is a need for effective malware detection systems.

Traditional malware detection systems use signature-based methods to identify malware. These methods rely on the presence of known malware signatures in the system. However, signature-based methods are not effective in detecting new or unknown malware samples. As a result, there is a need for more advanced malware detection systems that can identify new and unknown malware samples.

Machine learning, a subfield of artificial intelligence, has emerged as a promising approach to improve the accuracy and scalability of malware detection. Machine learning algorithms can automatically learn patterns and behaviors from large datasets without relying on predefined rules or signatures, making them well-suited for detecting unknown or zero-day malware. In recent years, there has been a growing body of research on malware detection using machine learning, with various techniques proposed and evaluated. Researchers have proposed various machine learning techniques for malware detection, including supervised and unsupervised learning algorithms.

Supervised learning algorithms, such as support vector machines (SVMs), decision trees, and artificial neural networks, have been used for malware detection. These algorithms are trained on a dataset of known malware and non-malware samples, and they learn to classify new samples as either malware or non-malware. However, these algorithms may struggle to detect new types of malware that have not been seen before.

Unsupervised learning algorithms, such as clustering and anomaly detection, have also been used for malware detection. These algorithms do not require labeled data and can detect unknown types of malware. However, they may also generate false positives, which can lead to high false alarm rates.

Deep learning algorithms have also been used for malware detection. These algorithms can learn complex patterns and relationships in data, making them well-suited for detecting malware. However, they require large amounts of data and computational resources to train.

In recent years, researchers have also proposed hybrid approaches that combine multiple machine learning techniques for malware detection. These approaches aim to improve the accuracy and robustness of malware detection systems. Overall, machine learning has shown great promise for malware detection, and it is likely to play a critical role in the development of effective malware detection systems in the future.

In this research paper, we provide a comprehensive review of the existing literature on malware detection using machine learning techniques. We discuss the different types of machine learning algorithms, feature extraction techniques, and evaluation metrics used in malware detection

research. We also highlight the challenges and limitations of using machine learning for malware detection and identify potential future directions for research in this field.

Literature Review

As per our proposed research, the prior state of art states that Yan et al. [1] offered a thorough survey on dynamic mobile malware detection approaches, summarizing a number of criteria and performance evaluation metrics for mobile malware detection. Additionally, the authors analyzed and compared the until then existing mobile malware detection systems based on the analysis methods and evaluation results. Finally, the authors pointed out open issues in the field and future research directions. Odusami et al. [2] surveyed mobile malware detection techniques in an effort to identify gaps and provide insight for effective measures against unknown Android malware. Their work showed that approaches which rely on ML to detect malicious apps were more promising and produced higher detection accuracy as opposed to signature-based techniques.

There were many computational approaches used by different authors, Kouliaridis et al. [3] provided a holistic review of works on the topic of mobile malware detection and categorized each of them under a unique classification scheme. Precisely, the latter groups each work based on its target platform, feature selection method, and detection techniques, namely signature-based or anomaly-based detection. Liu et al. [4] presented a comprehensive survey of Android malware detection approaches that utilize ML techniques. The authors analyzed and summarized several key topics, including sample acquisition, data preprocessing, feature selection, ML models, algorithms, and detection performance. Finally, they elaborated on the limitations of ML approaches and offered insights for potential future directions.

Gibert et al. [5] surveyed popular ML techniques for malware detection and in particular, deep learning techniques. The authors explained research challenges and limitations of legacy ML techniques and scrutinized recent trends and developments in the field with a focus on deep learning schemes. They categorized the surveyed works into three groups, namely static, dynamic, and hybrid. Shabtai et al. [6] contributed a system that detects malicious behavior through network traffic analysis. This is done by logging user-specific network traffic patterns per examined app and subsequently identifying deviations that can be flagged as malicious. To evaluate their model, they employed the C4.5 algorithm, achieving an accuracy of up to 94%.

Canfora et al. [7] suggested an Android malware detection scheme that analyzes op-code frequency histograms; this is accomplished by observing the frequency of occurrences of each group of op-codes. Precisely, their detection model capitalizes on a vector of features obtained from eight Dalvik op-codes. These op-codes are usually used to alter the app's control flow. Six classification models were employed during the evaluation, namely LadTree, NBTree,

RandomForest, RandomTree and RepTree. The proposed model was applied separately to the eight features and the three groups of features. The first group includes the move and the jump features, the second involves two well-known distance metrics, namely Manhattan and Euclidean distance, and the last embraces all the four features. The proposed method was evaluated on the Drebin dataset using several classifiers, namely J48, LadTree, NBTree, Random Forest, Random Tree and RepTree, and achieved an accuracy of 95%.

L. Li et al. [8] systematically reviewed the studies conducted in static analysis techniques used for Android applications from 2011 to 2015. The tools that can be used to perform Android code analysis using static analysis techniques were also summarized.Abstract representation, blob analysis, symbolic execution, program splitting, code instrumentation, and type/model checking were identified as core analytical methods. Though this review correctly identified the most widely used approach to detect privacy and security related issues, the applicability of static analysis techniques for malware detection was not discussed. Apart from that, it did not take into account the recent research where novel analysis methods and malware detection methods were suggested.

Y Pan et al. [9] provided a good systematic review mainly on static analysis techniques that can be used in Android malware detection. Four methods were identified based on characteristics, on opcodes, on program graphs and on symbolic execution. It then evaluated the static analysis capabilities of Android malware detection methods based on these four methods using existing literature. The paper identifies ML and statistical models as possible methods to identify Android malware. However, ML-based machine learning methods have not been thoroughly reviewed because the main focus is only on static analysis techniques. M.A. Ashawa et al.[10] said that there are five types of Android malware detection techniques. They are static and dynamic detection, hybrid detection, permission based detection, and emulation based detection. They also summarized the reviewed work with the model accuracy of malware detection, but the approach of those studies was not discussed.

T Sharma et al. [11] provides a good analysis of static, dynamic and hybrid detection techniques used in existing research studies for Android malware detection. Along with this possibility of using machine learning models, several deep learning models are also discussed. However, this study did not comprehensively analyze the model accuracy of machine learning methods for Android malware detection, as this study focused more on discussing different approaches to malware detection instead of considering the accuracy of these approaches. Thus, these works differ from our study.

S.M. Ghaffarian et al.[12] analyzed several studies on ML-based and data mining approaches that can be used to identify software vulnerabilities up to 2017. Although this survey provides a good analysis, they considered most of the research work in general software security. Therefore,

vulnerability analysis in Android code was not discussed. However, findings such as the use of ML models for vulnerability analysis are still beneficial for analysis related to specific programming languages.

Malware detection in Android is done in two ways. Signature-based detection methods and behavior-based detection methods [13]. Signature-based detection methods are simple, efficient, and produce fewer false positives. The program's binary code is compared to signatures using a database of known malware. However, it is not possible to detect unknown malware using this method. Therefore, behavior/abnormality based detection methods are the most common methods. This method usually involves techniques from machine learning and data science. detect Android malware using traditional ML-based methods such as Decision Trees (DT) and Support Vector Machines (SVM) and newer DL-based models such as Deep Convolutional Neural Network (Deep-CNN) [14].

Dataset Used

We are using the EMBER (Elastic Malware Benchmark for Empowering Researchers) dataset for our project and research purpose. The EMBER dataset is a collection of features from PE (Portable Executable) files that serve as a benchmark dataset for researchers. The EMBER2017 dataset contained features from 1.1 million PE files scanned in or before 2017 and the EMBER2018 dataset contains features from 1 million PE files scanned in or before 2018. This repository makes it easy to reproducibly train the benchmark models, extend the provided feature set, or classify new PE files with the benchmark models.

A labeled benchmark dataset for training machine learning models to statically detect malicious Windows portable executable files. The dataset includes features extracted from 1.1M binary files: 900K training samples (300K malicious, 300K benign, 300K unlabeled) and 200K test samples (100K malicious, 100K benign). Our aim is to do comparative analysis on this dataset using different machine learning and deep learning algorithms to derive new insights, which leads to optimized performance in an experiment.

The dataset being used for the project consist of 57 columns out of which 55 are numeric and 2 are categorical. The table shown below gives an idea about the type of features that exist in a portable executable file and which can be altered or changed for malware practices. These features are used to check the authenticity of the file and hence help to classify whether the file is malicious or not.

Name	Data Type	Description
MajorImageVersion	Int 64	MajorImageVersion is a characteristic of a Windows Portable Executable (PE) file, which represents the version of the

		operating system for which the file was originally designed. This characteristic is often used in malware detection to identify malicious files that have been designed for an older or newer version of the operating system than the one on the infected machine.
ResourcesMaxSize	Int 64	ResourcesMaxSize is a value that specifies the maximum size of the resource data in a Portable Executable (PE) file, which is a file format used for executables, object code, and DLLs in Windows operating systems. Malware can modify the ResourcesMaxSize value to increase the size of its payload or to evade detection by anti-malware tools that use the default value to identify legitimate executables
SizeOfStackReserve	Int 64	SizeOfStackReserve is a field in the Portable Executable (PE) header of a Windows executable file that indicates the amount of virtual memory to reserve for the program's stack. It specifies the maximum stack size in bytes that the operating system reserves for the program at load time.
SizeOfCode	Int 64	SizeOfCode is a field in the header of a Portable Executable (PE) file, which contains the size of the code section, in bytes, that the loader uses to create the process. In the context of malware, this field can be useful for analyzing the behavior of a malicious program, as it can give insight into the amount of code that is executed.
MinorLinkerVersion	Int 64	MinorLinkerVersion is a field in the Portable Executable (PE) file format, which is commonly used in Windows operating systems to store executable code, DLLs, and other binary files. It represents the minor version number of the linker used to create the file. This information can be useful in malware analysis to identify the toolset or development environment used to create the executable file

Table:1, Basic Features in Data Set

The EMBER dataset consists of a collection of JSON lines files, where each line contains a single JSON object. Each object includes the following types in data:

• The sha256 hash of the original file as a unique identifier;
• Coarse time information (month resolution) that establishes an estimate of when the file was first seen;
• A label, which may be 0 for benign or 1 for malicious; and
• Eight groups of raw features that include both parsed values as well as format-agnostic histograms.

Proposed Methodology

Malware, short for "malicious software," is a type of software that is intentionally designed to cause harm to a computer system, network, or device. Malware can take many forms, including viruses, worms, Trojans, ransomware, spyware, adware, and rootkits. Malware can be spread through various methods, such as email attachments, malicious websites, social engineering, or exploiting vulnerabilities in software or systems. Once malware infects a system, it can cause a range of problems, including data theft, system malfunction, loss of data, unauthorized access, and financial loss. Malware can also be used to launch attacks on other systems or networks, creating further damage and disruption.

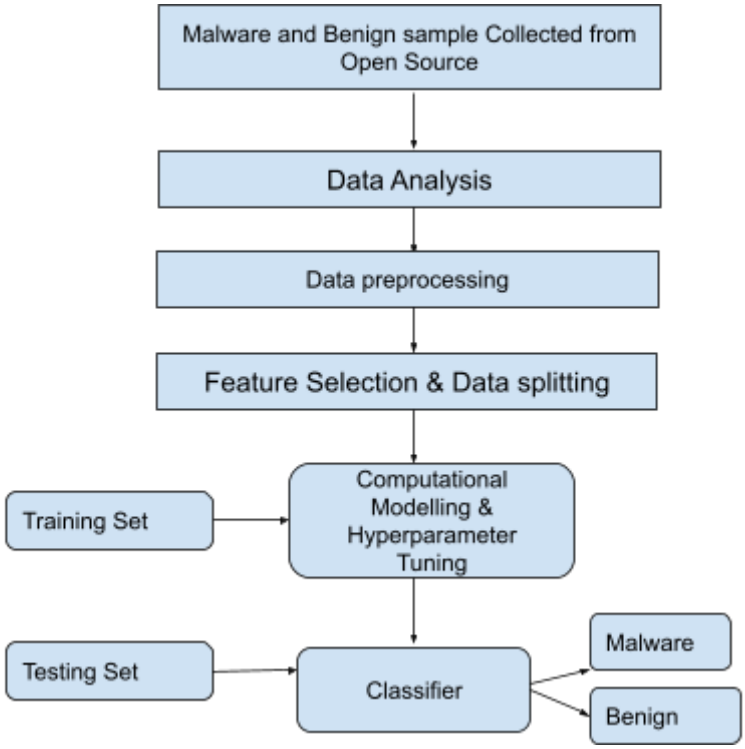

Figure:-1, Model Workflow

Preventing and detecting malware is an important part of cybersecurity. Antivirus software, firewalls, and other security measures can help protect against malware, while malware analysis and detection techniques such as signature-based detection, behavioral analysis, and machine learning can help identify and remove malware from infected systems.

In this paper, we propose a malware detection system using Machine Learning. The proposed system consists of two main components: a feature extractor and a deep neural network classifier. The feature extractor is used to extract the features of malware samples, and the classifier is used to classify the samples into different categories.

Data Pre-Processing

One hot encoding is a popular technique used to convert categorical data into a numerical representation that can be used in machine learning models. In one-hot encoding, each category is represented as a binary vector of size N, where N is the number of categories. Each vector has only one "1" at the index corresponding to the category it represents, and all other values are "0".

Standard scalar is a popular technique used in machine learning for feature scaling, which is the process of normalizing the range of input variables. Standard scalar scales the data to have a mean of zero and a standard deviation of one.

The standard scalar formula is as follows:

$$Z = \frac{(x - u)}{s}$$

where z is the standardized value,
x is the original value,
u is the mean of the feature, and
s is the standard deviation of the feature.

By using standard scalar, the features can be transformed to have a similar scale, which helps in improving the performance of machine learning algorithms. It is especially important when using algorithms that are sensitive to the scale of the input features, such as gradient descent-based algorithms.

Column Transformer is a scikit-learn class used to create and apply separate transformers for numerical and categorical data. To create transformers we need to specify the transformer object and pass the list of transformations inside a tuple along with the column on which you want to apply the transformation.

```
[16]
numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder()

[17]
preprocessor = ColumnTransformer(
    [
        ("OneHotEncoder", oh_transformer, cat_features),
        ("StandardScaler", numeric_transformer, num_features),
    ]
)
```

Figure:-2, Implementation of One-Hot Encoder & Standard Scaler

Feature selection, data splitting & modeling

- *Sklearn:*

Scikit-learn (also known as sklearn) is a popular open-source machine learning library for Python. It provides a wide range of tools for data preprocessing, feature extraction, and model selection, as well as a variety of machine learning algorithms. Sklearn can be used in malware detection to build machine learning models that can automatically classify files as either malicious or benign. Sklearn provides several machine learning algorithms that are commonly used in malware detection, including decision trees, random forests, and support vector machines (SVM). Sklearn also provides tools for data preprocessing and feature extraction, which are essential steps in building machine learning models for malware detection. Sklearn provides tools for scaling, normalizing, and transforming data, as well as feature extraction techniques such as principal component analysis (PCA) and discrete wavelet transform (DWT).

In addition, Sklearn provides tools for model selection and evaluation, which are important for ensuring that the machine learning model is robust and accurate. Sklearn provides tools for cross-validation, grid search, and hyperparameter tuning, as well as metrics for evaluating the performance of the model, such as precision, recall, and F1 score.

Sklearn is widely used in malware detection research and has been used to build several effective machine learning models for detecting malware. However, it should be noted that machine learning models for malware detection are not foolproof and can be circumvented by advanced malware techniques. Therefore, it is important to use a combination of different techniques, including machine learning, signature-based detection, and behavioral analysis, to ensure robust malware detection.

- *Matplotlib:*

Matplotlib is a widely used data visualization library for Python that provides a range of tools for creating various types of visualizations, such as scatter plots, histograms, and heatmaps. In the context of malware detection, Matplotlib can be used to visualize the features of malware and benign files and gain insights into the data, which helps developers to build better machine learning models for malware detection. Matplotlib can be used to visualize the distribution and relationship between features of malware and benign files. For example, scatter plots can be used to visualize the relationship between two features, such as file size and entropy. A scatter plot can show if there is a correlation between the two features, and if there is a clear boundary between the malware and benign files in the plot.

Histograms can be used to visualize the distribution of a single feature, such as byte frequency. A histogram can show if there is a clear separation between the distributions of malware and benign files, which can be used as a feature for machine learning models. Heatmaps can be used

to visualize the correlation between multiple features. A heatmap can show if there are groups of features that are highly correlated, which can be used to reduce the dimensionality of the feature space or identify redundant features.

In addition to visualizing features, Matplotlib can be used to visualize the performance of machine learning models for malware detection. For example, Matplotlib can be used to create confusion matrices, ROC curves, and precision-recall curves, which can be used to evaluate the performance of the machine learning models.

Overall, Matplotlib is a powerful and widely used data visualization library that can be used in malware detection to gain insights into the data and visualize the performance of machine learning models.

- *Seaborn:*

Seaborn is a data visualization library for Python that is built on top of Matplotlib. Seaborn provides a range of tools for creating various types of statistical graphics, such as scatter plots, line plots, bar plots, and heatmaps. In the context of malware detection, Seaborn can be used to visualize the features of malware and benign files and gain insights into the data, which can help developers to build better machine learning models for malware detection.

Seaborn can be used to visualize the distribution and relationship between features of malware and benign files. For example, Seaborn provides tools for creating scatter plots with regression lines and confidence intervals, which can help to identify linear relationships between variables. Seaborn can also be used to create kernel density estimates (KDEs), which can be used to estimate the probability density function of a feature. Seaborn can also be used to create heatmaps with clustered rows and columns, which can help to identify groups of features that are highly correlated. Seaborn provides tools for hierarchical clustering, which can be used to cluster the features based on their correlation. This can help to identify groups of features that are redundant or highly correlated, which can be used to reduce the dimensionality of the feature space or identify important features.

In addition to visualizing features, Seaborn can be used to visualize the performance of machine learning models for malware detection. For example, Seaborn provides tools for creating confusion matrices, ROC curves, and precision-recall curves, which can be used to evaluate the performance of the machine learning models.

Overall, Seaborn is a powerful and widely used data visualization library that can be used in malware detection to gain insights into the data and visualize the performance of machine learning models. It provides a range of statistical graphics and tools for clustering and visualizing

high-dimensional data, which can be used to identify important features and reduce the dimensionality of the feature space.

- *Catboost:*

CatBoost is an open-source gradient boosting library that is designed to work with high-dimensional categorical data. It is developed by Yandex, a Russian technology company, and is available for use in Python, R, and command-line interfaces.

In the context of malware detection, CatBoost can be used to build machine learning models that can automatically classify files as either malicious or benign based on a set of features extracted from the files. CatBoost is particularly well-suited for malware detection because it can handle high-dimensional categorical data, which is commonly used in malware analysis.

CatBoost uses gradient boosting algorithms to build machine learning models that are based on an ensemble of decision trees. The model is built iteratively, with each iteration adding a new decision tree that is trained to correct the errors of the previous iteration. The gradient boosting algorithm also includes a regularization parameter that helps to prevent overfitting and improve the generalization performance of the model.

In addition to its ability to handle categorical data, CatBoost provides several other features that make it well-suited for malware detection. These include:

1. Feature importance: CatBoost provides a feature importance measure that can be used to identify the most important features for malware detection. This can help researchers and practitioners to focus on the most informative features and reduce the dimensionality of the feature space.
2. Hyperparameter tuning: CatBoost provides several hyperparameters that can be tuned to optimize the performance of the model, such as the number of trees, the learning rate, and the regularization parameter. CatBoost provides tools for hyperparameter tuning, such as grid search and random search, which can help to find the optimal values for these parameters.
3. Interpretability: CatBoost provides tools for interpreting the decisions of the model, such as feature contributions and SHAP values. This can help to understand the behavior of the model and identify potential vulnerabilities or biases.

Overall, CatBoost is a powerful and widely used machine learning library that can be used in malware detection to handle high-dimensional categorical data and build accurate and interpretable models.

- *XGBoost:*

XGBoost is an open-source library that implements gradient boosting algorithms for machine learning. It is designed to work with both numerical and categorical data and is highly scalable and efficient. In the context of malware detection, XGBoost can be used to build machine learning models that can classify files as either malicious or benign based on a set of features extracted from the files.

XGBoost is particularly well-suited for malware detection because it can handle high-dimensional data and is highly efficient in terms of both memory usage and computation time. This makes it possible to build models on large datasets with thousands of features and millions of samples.

XGBoost uses a gradient boosting algorithm to build an ensemble of decision trees that are trained iteratively to minimize a loss function. The algorithm works by adding new trees to the ensemble that are trained to correct the errors of the previous trees. The optimization is done using a gradient descent algorithm that updates the weights of the samples based on the errors of the previous iteration.

In addition to its ability to handle high-dimensional data and its efficiency, XGBoost provides several other features that make it well-suited for malware detection. These include:

1. Regularization: XGBoost provides several regularization techniques, such as L1 and L2 regularization and early stopping, that can help to prevent overfitting and improve the generalization performance of the model.
2. Feature importance: XGBoost provides a feature importance measure that can be used to identify the most important features for malware detection. This can help researchers and practitioners to focus on the most informative features and reduce the dimensionality of the feature space.
3. Hyperparameter tuning: XGBoost provides several hyperparameters that can be tuned to optimize the performance of the model, such as the number of trees, the learning rate, and the regularization parameter. XGBoost provides tools for hyperparameter tuning, such as grid search and random search, which can help to find the optimal values for these parameters.

Overall, XGBoost is a powerful and widely used machine learning library that can be used in malware detection to handle high-dimensional data, improve the accuracy and generalization performance of the model, and identify the most important features for malware detection.

- *Random Forest Regressor:*

Random Forest Regressor is an ensemble machine learning algorithm that is commonly used for regression tasks. It works by building a collection of decision trees, where each tree is built on a random subset of the training data and a random subset of the features. The algorithm works by taking the average of the output from all the decision trees in the forest to make a final prediction. Because it is an ensemble model, it is generally more accurate than a single decision tree.

One of the main advantages of Random Forest Regressor is that it is less prone to overfitting compared to other machine learning models. This is because the algorithm averages the output from multiple decision trees, which helps to reduce the variance in the model. Another advantage is that the algorithm can handle both numerical and categorical data, and can also handle missing values. It is also relatively easy to use and can be implemented using existing libraries in Python such as scikit-learn. However, Random Forest Regressor is not without its limitations. The main disadvantage is that it can be slow to train on large datasets, especially if the number of trees in the forest is large. It also requires more memory to store the forest compared to a single decision tree.

In addition to its ability to handle high-dimensional data and its robustness to noisy and irrelevant features, Random Forest Classifier provides several other features that make it well-suited for malware detection. These include:

1. Outliers: Random forest regressors are generally robust to outliers, but extremely influential outliers can still affect the accuracy of the model. Consider using techniques such as Winsorization or trimming to deal with outliers.
2. Feature importance: Random forest regressors can provide insight into the relative importance of each feature in the prediction task. Consider using the feature_importances_ attribute of the fitted model to gain insights into which features are most important.
3. Hyperparameter tuning: Random forest regressors have several hyperparameters that can be tuned to improve their performance. Consider using techniques such as grid search or random search to explore different combinations of hyperparameters.
4. Model interpretability: Random forest regressors can be difficult to interpret due to their ensemble nature. Consider using techniques such as partial dependence plots or permutation feature importance to gain insights into how the model is making predictions.

5. Data quality: As with any machine learning model, the quality of the input data can greatly affect the accuracy of the model. Consider performing exploratory data analysis and data cleaning to ensure that the input data is of high quality.
6. Overfitting: Random forest regressors can be prone to overfitting, especially when the number of trees is large. Consider using techniques such as cross-validation or early stopping to prevent overfitting.

Overall, Random Forest Classifier is a powerful and widely used machine learning algorithm that can be used in malware detection to handle high-dimensional data, reduce overfitting, and identify the most important features for malware detection. Comparing ML models for a project is important because it helps in identifying the best performing model for a given dataset and problem. It also provides insights into the strengths and weaknesses of different models and helps in selecting the most suitable model for deployment.

By comparing multiple ML models, we can evaluate their performance on various metrics such as accuracy, precision, recall, F1 score, and others. We can also use techniques such as cross-validation and hyperparameter tuning to ensure that the models are being evaluated fairly and that their parameters are optimized for the given dataset.

Moreover, comparing ML models can also help in understanding the underlying relationships between the features and the target variable, and can provide insights into how different algorithms handle different types of data. This knowledge can be valuable for future projects and can help in selecting the most appropriate ML model for a given task.

HyperParameter Tuning

Hyperparameter tuning is the process of selecting the optimal hyperparameters for a machine learning model. Hyperparameters are values that are set before training a model and cannot be learned during training. They have a significant impact on the performance of a model and hence, selecting the right values for hyperparameters is important to achieve the best possible performance.

Hyperparameter tuning involves selecting the optimal values for hyperparameters through various techniques such as Grid Search, Random Search, Bayesian Optimization, etc. Grid Search involves defining a grid of hyperparameters and testing each combination of hyperparameters to find the optimal one. Random Search involves randomly sampling values for hyperparameters from a defined distribution and testing them to find the optimal values. Bayesian Optimization involves using a probabilistic model to search for optimal values.

Hyperparameter tuning can significantly improve the performance of a model and help to avoid overfitting or underfitting of the model. It can also help to reduce training time and cost by selecting the optimal hyperparameters. Here we are only using randomizedsearchCV and gridsearchCV for the research purpose.

RandomizedSearchCV is a technique used for hyperparameter tuning in machine learning models. It is used to search over a specified hyperparameter space for the best combination of hyperparameters that optimize a specified scoring metric.

The process involves randomly selecting hyperparameter combinations from the specified hyperparameter space and evaluating the performance of the machine learning model on a validation set. This process is repeated a specified number of times, or until a specified computational budget is exhausted.

The benefits of using RandomizedSearchCV over a traditional grid search approach include faster computation time and a higher likelihood of finding a better set of hyperparameters. In grid search, every possible combination of hyperparameters is tested, which can be computationally expensive, especially for large hyperparameter spaces. RandomizedSearchCV allows for a more efficient search by randomly sampling hyperparameters from the specified search space. Once the best set of hyperparameters is found, the machine learning model can be trained on the entire training set using these hyperparameters to make predictions on new data.

GridSearchCV works by exhaustively searching over a specified parameter grid and fitting the model on each combination of hyperparameters to find the best combination that gives the highest score on a specified evaluation metric.

The grid of hyperparameters to be searched is specified as a dictionary with keys as the hyperparameters and values as a list of values to be tried for each hyperparameter. GridSearchCV then trains the model on each combination of hyperparameters using cross-validation and returns the combination that results in the highest score on the specified evaluation metric.

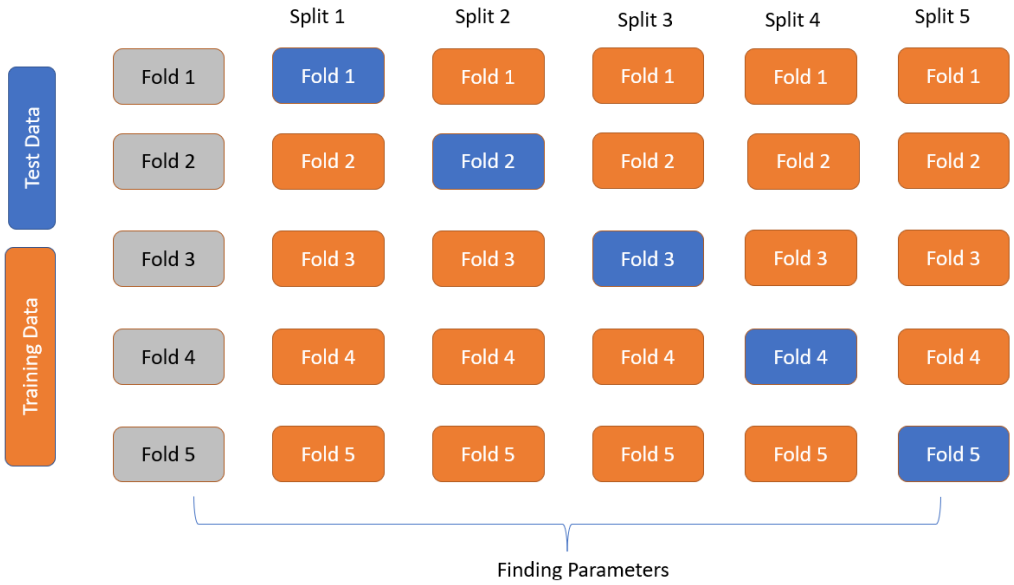

Figure:-3, Cross-Validation Grid Search

It is recommended to use Grid Search after Randomized Search CV because Randomized Search CV can be used to narrow down the range of hyperparameters that need to be searched, which can save a lot of computation time. Randomized Search CV selects random combinations of hyperparameters from the provided range and evaluates the model with each combination. This approach is efficient when there are a large number of hyperparameters to search and the range of values is wide.

Once the range of hyperparameters is narrowed down using Randomized Search CV, Grid Search can be used to perform a more focused search within the narrowed-down range. Grid Search exhaustively searches all the possible combinations of hyperparameters within the specified range. This approach is useful when the number of hyperparameters to search is smaller and the range of values is narrow.

By using Randomized Search CV followed by Grid Search, we can strike a balance between the efficiency of the search and the thoroughness of the search, leading to a better chance of finding the optimal set of hyperparameters for our model.

Deep Learning Algorithm

Malware detection using deep learning ANN models has shown promising results in recent research. Deep learning models, such as neural networks, are well-suited for detecting malware because they can learn complex patterns and relationships in the data, which is important for identifying the subtle differences between malware and benign software.

One advantage of deep learning models is that they can handle large and complex datasets, which is essential for malware detection. Deep learning models can also be trained on a variety of features, including static and dynamic analysis features, which are commonly used in malware detection.

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 64)                1352000

 dense_13 (Dense)            (None, 32)                2080

 dense_14 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,354,113
Trainable params: 1,354,113
Non-trainable params: 0
_________________________________________________________________
```

Figure:-4, Model Summary

Binary cross entropy is a commonly used loss function for binary classification tasks, such as malware detection. It measures the difference between the predicted probability distribution and the actual distribution of the target variable. Specifically, it calculates the cross-entropy loss between the predicted probability distribution and the actual distribution of the target variable. In the case of binary classification, this distribution is simply a probability value indicating the likelihood of a sample belonging to one of the two classes.

Adam optimizer, on the other hand, is an adaptive learning rate optimization algorithm used for gradient-based optimization of deep learning models. It combines the advantages of two other popular optimization algorithms, AdaGrad and RMSProp, to improve the learning process. Adam optimizer is known for its ability to converge quickly and handle sparse gradients efficiently, making it a popular choice for training deep learning models.

In the context of malware detection using ANN, binary cross entropy is used as the loss function to measure the difference between the predicted and actual classification of malware and non-malware samples. Adam optimizer is used to optimize the weights of the ANN to minimize this loss function, resulting in a model that can accurately detect malware samples with high precision and recall.

Experimental Results

Throughout the research we have performed various algorithms of Machine Learning & Deep learning but after comparative analysis, we observed that the best result was obtained from XGBoost Regressor, Random Forest Regressor, and Cat Boosting Regressor. Later, we have Fine-tune the model by adjusting hyperparameters & implemented Artificial Neural Network in order to achieve better performance & hence it resulted in 96.24% accuracy.

S.No.	Model Name	Accuracy
1	XGBoost Regressor	92.85%
2	Random Forest Regressor	92.64%
3	Cat Boosting Regressor	92.51%
4	K-Neighbors Regressor	90.17%
5	Decision Tree	89.66%
6	AdaBoostRegressor	79.89%
7	Lasso	-0.010%
8	Ridge	-75.19%

Table:2, Comparative Result Analysis

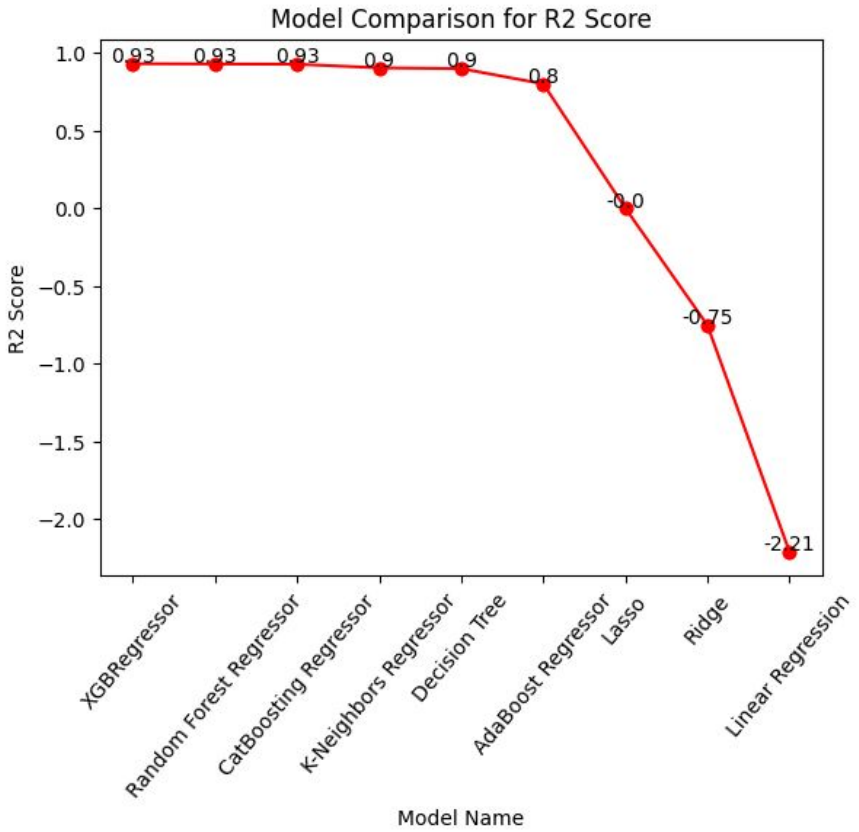

Figure:-5, Model Result Overview

Conclusion

This study investigated the effectiveness of machine learning techniques for malware detection. XGBoost Regressor showed the best result with an accuracy of 92.85%. Followed by Random Forest Regressor and CatBoosting Regressor with an accuracy of 92.64% and 92.51%. Other machine learning algorithms did not flair well while Lassos and Ridge regression performed the worst. The results showed that deep learning models, such as the ANN model with binary cross entropy loss and Adam optimizer, outperformed traditional machine learning algorithms with an accuracy of 96%. These findings have significant implications for the development of more effective malware detection systems.

However, this study had some limitations. Firstly, the dataset used was limited in size and diversity, which may limit the generalizability of the results. Secondly, the study only focused on one type of malware and did not consider other forms of malware. Future research should address these limitations by using larger and more diverse datasets and by considering different types of malware.

Overall, this study contributes to the growing body of research on malware detection using machine learning techniques. The findings suggest that deep learning models have the potential to improve the accuracy of malware detection, which can enhance the security of computer systems. Future research could explore the use of other machine learning algorithms and data preprocessing techniques to further improve the accuracy of malware detection systems. Ultimately, the development of more effective and reliable malware detection systems will require ongoing research and collaboration between experts in the fields of cybersecurity and machine learning.

Future Work

One of the main ideas behind using adversarial machine learning for malware detection is to train a model that is robust against adversarial attacks. This means that the model can still accurately detect malware even if it has been modified or obfuscated to evade detection.

One approach is to use a deep neural network to classify malware samples, and then use adversarial training to improve the robustness of the classifier. Adversarial training involves generating perturbed versions of the original malware samples that are designed to fool the classifier, and then re-training the classifier on both the original and perturbed samples. This process can help the classifier to better distinguish between malicious and benign samples. The approach to implementing adversarial machine learning for malware detection is to use generative adversarial networks (GANs). GANs are composed of two neural networks: a generator and a discriminator. The generator is trained to generate new malware samples that are

similar to real malware samples, while the discriminator is trained to distinguish between real and fake malware samples.

Another approach is to use explainable AI techniques to better understand the features that the classifier is using to make its predictions. This can help to identify which features are most important for detecting malware, and can also help to identify cases where the classifier may be making incorrect predictions. To implement explainable AI for malware detection, techniques such as LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (Shapley Additive Explanations) can be used. These techniques can help to provide insights into why a particular malware sample was classified as malicious or benign by the model. This can help to increase trust in the model's decisions and improve its overall effectiveness.

Reference:

[1] Yan, P.; Yan, Z. A survey on dynamic mobile malware detection. *Softw. Qual. J.* **2017**, *26*, 891–919.DOI: https://doi.org/10.1007/s11219-017-9368-4

[2] Odusami, M.; Abayomi-Alli, O.; Misra, S.; Shobayo, O.; Damasevicius, R.; Maskeliunas, R. Android Malware Detection: A Survey. In *Communications in Computer and Information Science*; Springer International Publishing: New York, NY, USA, 2018; pp. 255–266. DOI: https://doi.org/10.1007/978-3-030-01535-0_19

[3] Kouliaridis, V.; Barmpatsalou, K.; Kambourakis, G.; Chen, S. A Survey on Mobile Malware Detection Techniques. *IEICE Trans. Inf. aSyst.* **2020**, *E103.D*, 204–211. DOI: 10.1587/transinf.2019INI0003

[4] Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access* **2020**, *8*, 124579–124607. DOI: 10.1109/ACCESS.2020.3006143

[5] Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. DOI: https://doi.org/10.1016/j.jnca.2019.102526

[6] Shabtai, A.; Tenenboim-Chekina, L.; Mimran, D.; Rokach, L.; Shapira, B.; Elovici, Y. Mobile malware detection through analysis of deviations in application network behavior. *Comput. Secur.* **2014**, *43*, 1–18. DOI: https://doi.org/10.1016/j.cose.2014.02.009

[7] Canfora, G.; Mercaldo, F.; Visaggio, C.A. Mobile Malware Detection using Op-code Frequency Histograms. Proceedings of the 12th International Conference on Security and Cryptography, SCITEPRESS—Science and and Technology Publications, Colmar, France, 20–22 July 2015. DOI: 10.5220/0005537800270038

[8] Li, L.; Bissyandé, T.F.; Papadakis, M.; Rasthofer, S.; Bartel, A.; Octeau, D.; Klein, J.; Traon, L. Static analysis of android apps: A systematic literature review. Inf. Softw. Technol. 2017, 88, 67–95. DOI: https://doi.org/10.1016/j.infsof.2017.04.001

[9] Pan, Y.; Ge, X.; Fang, C.; Fan, Y. A Systematic Literature Review of Android Malware Detection Using Static Analysis. IEEE Access 2020, 8, 116363–116379. DOI: 10.1109/ACCESS.2020.3002842.

[10] Ashawa, M.A.; Morris, S. Analysis of Android malware detection techniques: A systematic review. Int. J. Cyber-Secur. Digit. Forensics 2019, 8, 177–187. DOI: https://doi.org/10.17781/P002605

[11] Sharma, T.; Rattan, D. Malicious application detection in android—A systematic literature review. Computer Sci. Rev. 2021, 40, 100373. DOI: https://doi.org/10.1016/j.cosrev.2021.100373

[12] Ghaffarian, S.M.; Shahriari, H.R. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Comput. Surv. (CSUR) 2017, 50, 1–36. DOI: https://doi.org/10.1145/3092566

[13] Chen, T.; Mao, Q.; Yang, Y.; Lv, M.; Zhu, J. TinyDroid: A lightweight and efficient model for Android malware detection and classification. Mob. Inf. Syst. 2018, 2018.DOI: https://doi.org/10.1155/2018/4157156

[14] Nisa, M.; Shah, J.H.; Kanwal, S.; Raza, M.; Khan, M.A.; Damaševičius, R.; Blažauskas, T. Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolutional neural network features. Appl. Sci. 2020, 10, 4966. DOI: https://doi.org/10.3390/app10144966