National College of Ireland

Malware Detection in Executable files using XGBoost Algorithm

MSc Academic

Cybersecurity

Yashvardhan Pant
Student ID: - x21132399

School of Computing
National College of Ireland

Supervisor: - Prof. Imran Khan

National College of

Ireland MSc Project

Submission Sheet

School of Computing

Student Name:	Yashvardhan Pant		
Student ID:	21132399		
Programme:	MSc in Cyber Security	**Year:**	2022-2023
Module:	MSc Research Project		
Supervisor:	Mr. Imran Khan		
Submission Due Date:	15/12/2022		
Project Title:	Malware Detection in Executable files using XGBoost Algorithm		
Word Count:	Count 5615	Page 19	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature:	Yashvardhan Pant
Date:	15/12/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Malware Detection in Executable files using XGBoost Algorithm

Yashvardhan Pant

X21132399

Abstract

Keeping up with the fast growth of modern technology has led to an increase in malware and harmful activities, which hackers are using to their advantage in stealing personal information and login passwords. Rising numbers of malicious software regularly attacking online systems now pose a significant risk. Due to the rapid growth of malware, manual heuristic inspection of samples is no longer regarded a viable method of analysis. The use of machine learning for automated, behavior-based malware detection and static approach with PE-header based malware detection information is thus seen as a powerful approach. In this study, examination of many Machine Learning Algorithms (such as XGBoost, KNN, and Random Forest) that may be used to detect malware by analyzing a large dataset. Here, measurement of how well things is doing utilizing a confusion matrix and a focus on Detection. XGBoost has the highest accuracy and precision of any method, with 98.292% and 99.15%, without feature extraction and 98.33% and 99.01% with feature extraction respectively.

Keywords: - Malware Detection, Machine Learning, Deep learning, Security, Algorithm, Dataset, XGBoost.

1. INTRODUCTION

Malware, which is short for "malicious software," is designed to do damage by destroying data that the user needs, compromising the security of the system by lurking there undetected, controlling the user's computer as if it were a robot and making the user a conduit for criminal activity, and so on. Researchers are working on several different methods for preventing and identifying malware. Automatic dynamic (behavior) malware analysis mixed with data mining activities, such as machine learning (classification) algorithms, is one recommended methodology (solution) for achieving efficacy and efficiency in identifying malware.

For malware identification, several researchers [7] turned to static analysis while the others relied on dynamic analysis. Static analysis, also known as signature-based analysis, is an evaluation of source code that doesn't involve running the program. Malware may be detected through dynamic analysis by seeing how a program acts after being executed in a virtual environment. Due to the specialized nature of dynamic analyses, they are more expensive and need more resources to be carried out. Static analysis is preferable to dynamic analysis because it is more stable and secure. Computers have gotten increasingly complicated and advanced because of technological development. Malware has also evolved into a form that is difficult to detect. To combat the inherent false alarm rate of behavior-based detection, a new method called "feature detection" was developed. It seemingly recognizes certain attack patterns, this method involves monitoring program executions and detecting deviations from

the defined in the behavior of the program. This method is quite like anomaly detection; however, rather of depending on machine learning approaches, it relies on characteristics that have been defined by humans to capture the behavior of the system.

In this suggested model, static analysis is used. The model splits into four distinct stages: data collection and processing, model development and validation, model predictions, and model effectiveness. This work offers a simple, fast, and accurate malware detection method that utilizes information extracted from the PE header and Section table to categorize malware families [10]. The system may identify previously unnoticed malicious code in the executable. The PE file format is the primary format for Windows executable files. There are many applications for it. Sample collection, feature extraction, dataset partitioning, and executable file categorization are the four fundamental pillars of the model. It scans executable files for malicious code before they are run. Malware analysis makes use of a variety of classification techniques based on supervised machine learning to determine whether an executable file was created maliciously. Precision, recall, accuracy, true positive rate, and false positive rate are only few of the measures used to evaluate how well the suggested system performs.

Below is the outline for the rest of the paper: In part 2, provision several works in progress and briefly review various malware detection methods that have been developed. Section 3 explains the methodology behind the suggested model. The findings of the experiments and a comparison are reported in Section 4, followed by a discussion of the results in Section 5, and finally a conclusion is offered in Section 6.

Research question: How can we leverage machine leaning algorithms to classify malignant and benign executable files

2. LITERATURE REVIEW

This research aims to propose a machine learning strategy for detecting malware. The executable files on Windows are the primary topic of this research. To keep up with the exponential proliferation of malicious software, there is a need to use a variety of automated methods for identifying infected files. This project will throw some light on utilization of a script that extracts information from PE-files so that machine learning training of algorithms on a dataset consisting of both infected and uninfected files.

An in-depth literature assessment of other researchers' efforts in malware detection is explained in the connected text. The following section elaborates on the study's methodology, tactics, and technical considerations, as well as the researchers' findings and potential future applications. Researchers have previously suggested several different methodologies and frameworks for identifying malicious software on the PC platform. The suggested study takes their findings and any limitations in their studies into account.

It was Schultz et al. [9] who first looked at the effectiveness of ML methods in identifying malicious software. To extract static characteristics, they used three distinct strategies. The first technique is called binary profiling, and it involves generating three distinct binary vectors for each binary, including a list of the Dynamically Linked Libraries (DLL) that the binary called, a list of the functions that have been called within each DLL, and the number of functions called within each DLL. Aside from byte sequence, the other two methods of

feature extraction were strings. The binary profile features were used to train a RIPPER rule-based learner, a Naive Bayes classifier was used to classify the binary strings, and an ensemble classifier was used to classify the binary byte sequences. The ensemble classifier had the greatest detection rate (97.76%) of the three learning methods evaluated, and its rate was twice that of conventional signature-based detection systems.

Firdausi, et al., [2] researchers' in-depth study on the use of machine learning for automated behavior-based malware detection has been completed, and the results are promising. Malware activity in an emulation environment may be automatically examined, leading to behavior reports. As many as 220 distinct pieces of malware were obtained. They applied several ML algorithms to the same collection of dynamic reports produced by Anubis Sandbox and incorporated in vector space models, and then compared their detection rates. k-Nearest Neighbors (KNN), Naive Bayes (NB), J48 Decision Tree (J48DT), Support Vector Machine (SVM), and Multilayer Perceptron Neural Network (MLPNN) were the techniques used (MLPNN). When everything was said and done, the J48 decision Tree's 96.8% accuracy was highlighted as the best detection rate. Initially, these reports will be used to build sparse vector models for machine learning (classification). After doing tests on all 5 classifiers, J48 decision tree came out on top in terms of recall (95.7 percent), false positive rate (2.4 percent), precision (97.3 percent), and accuracy (96.8 percent).

Malware detection using machine learning algorithms might be very efficient and effective. Using a static analysis method, Radwan et al. [8] presented a system for extracting characteristics of PE files. There were seven distinct categorization algorithms utilized. The dataset was broken down into its constituent parts: the raw features, which had 53 features, and the integrated features, which included both derived and enlarged features. There are 74 of them in all. When compared to other algorithms on integrated feature datasets, random forest fared best with 93.23% accuracy on the 70/30 split and 97.56% accuracy on the raw dataset. Both the K-Nearest Neighbor (KNN) model and the Gradient Boosting (GB) tree fared well with tenfold cross-validation on the combined dataset, with an accuracy of 98.70% and 93.55%, respectively.

Sandbox, feature extractor, and classifier were all built by Wang et al. [13]. Their process consisted of three distinct steps: collection, analysis, and classification. The collector has both a static analysis program and the PinFWSandbox module for dynamic execution. It took note of data changes in dynamic files and log files and sent those data on to the extractor phase. The extractor might extract features in a static, dynamic, or system call manner. The last component was a classifier. When the results of all classifiers were averaged together, it was discovered that the dynamic op-code classifier produced the highest F1-score (96%) of all classifiers.

To identify PE-type malware in Windows, Catak et al. [1] leveraged the of operating system API calls, which is an interesting and potentially useful job. An official definition of this work is to execute malware in a sandbox, log API calls made by Windows, and then sequentially analyze the logs. First, the contents of the suspicious software are examined without running the software, and then the software is launched in an isolated environment, explored for DNS resolution requests, and the registry, file system, and API calls are recorded. They isolated 7107 pieces of dangerous software (viruses, backdoors, and trojans)

in a sandbox environment and examined them, then converted their findings into a format that can be read by various categorization algorithms and techniques.

The use of ML for Windows malware detection was the subject of a review that Naz et al. [7] Using a classifier to train the model on extracted features of the PE file, they introduced a static analysis method based on machine learning; however, the dataset used was relatively small, resulting in average accuracy; they achieved the highest accuracy (97.24 percent) with the random forest classifier. The best accuracy of 98.63 percent was attained by a random forest classifier when the model was trained and evaluated on a dataset consisting of extracted features from the file, the optional, and the section header.

To accurately detect unknown malware, Darshan et al. [10] suggested a hybrid technique that combined static and dynamic aspects of PE files with a linear support vector classification strategy. Training the algorithm on a limited dataset hampered attempts to improve accuracy. Experimental findings using tenfold cross-validation show that the proposed HF-MDS is effective at accurately identifying malware and benign PE files, with a detection accuracy of 99.743% achieved using a hybrid features-based sequential minimum optimization classifier.

Galal et al. [3] developed a model of behavioral traits to characterize the malicious behavior of malware. The suggested approach is derived from dynamic analysis of a recent malware dataset conducted in a simulated, controlled environment to collect traces of API calls made by malicious programs. Once the traces are collected, they are aggregated into more abstract characteristics call actions. Multiple categorization techniques, including the decision tree, random forests, and support vector machine, are used to evaluate the efficacy of potential courses of action. When compared to API n-gram based approaches, the semantic value of describing dangerous activities via actions is much higher. Malware analysts may also put their technical expertise to use by extending the current set of heuristic methods to extract new behaviors. The suggested feature model achieved an accuracy of 97.19% with Decision Tree and a lower accuracy of 95% using SVM.

Kolter et al. [6] detecting and classifying malware in production environments. The author asserts that 2,012 training samples were employed, including 1,971 clean files and 1,651 malicious executables encoded using n-gram characteristics. After selecting the features, the authors claim to have utilized a variety of evaluation methods such as Naive Bayes, Decision trees, Support vector machines, and boosting. The authors observed their technique classifying programs according to payload and found it to be scalable to a large number of executables. All the files were in the window PE format; the clean files came from the home directories of PCs running Windows XP and Windows 2000, while the malicious files were all caught on the network. The executables were "hex dumped," or converted to hexadecimal code using the ISO-8859-1 character set, so that computers could read them. After then, the famed n-grams were created by stringing together each four-byte sequence into a single phrase. Experiments show that a boosted decision tree can accurately identify harmful files 98% of the time, with only 6% of false positives for dangerous files. While this may seem like a major issue to some, the author argues that if the reader is ready to tolerate a false positive rate of 0.1, then the detection rate will be 100%. The author concludes that boosted J48 is the most effective algorithm based on its ROC curve of 0.996, outperforming the others assessed.

Fewer studies have examined the XGBoost classifier as a Machine Learning Algorithm, despite the fact that researchers are consistently striving to improve the accuracy and precision of their datasets by employing various static methods, such as PE header information, or behavioral ones, such as API, files deleted, rekeys etc. of the executables during their running time. J48, KNN, SVM, and RF are the common methods used in study. Researchers also use Deep Neural networks to determine if a file contains cancerous cells or not. When comparing several algorithms, XGBoost machine learning models provide the most optimal trade-off between prediction accuracy and speed. Feature extraction has an impact on the accuracy and precision of the models used for executable categorization; this study compares XGBoost to other machine learning techniques and examines this impact.

3. METHODOLOGY

Sample collection, feature extraction, and classification are the three main pillars of this research methodology. Various samples of executable applications from different files are collected and labeled before their runtime behavior is generated in the Cuckoo sandbox and features are extracted from there.

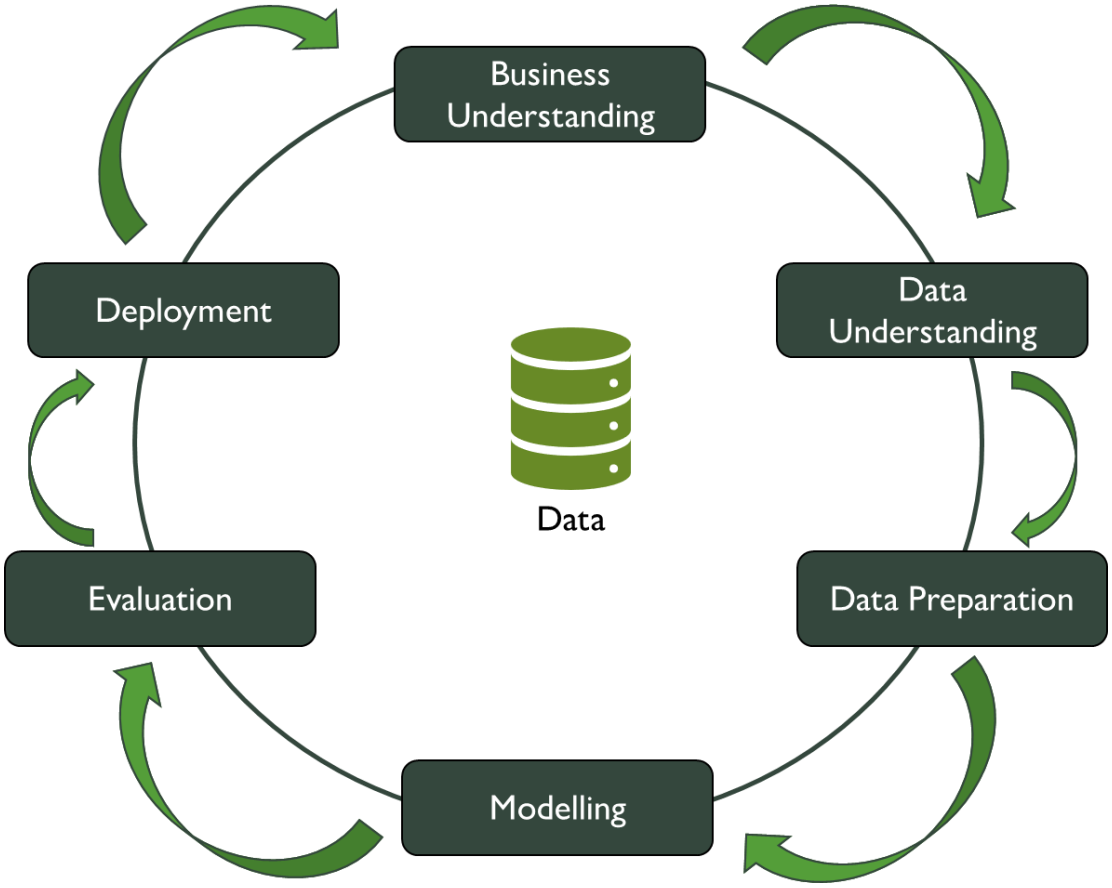

Figure 1: CRISP-DM Model

During the research utilization of CRISP-DM (Common Industry Data Mining Process) is both implement and assessed in the model [4]. It's a procedure for data mining that can be used to any business, and it offers a great framework for organizing your data mining project.

The Data Science team at various companies has found remarkable success with this technique, which has its origins at IBM and was designed for Data Mining. The phases involved in the CRISP-DM technique are outlined below the graphic, which depicts the process of the CRISP-DM approach.

3.1 Business Requirement:

There is a growing security risk due to the proliferation of malicious software that exploits online systems every day. Due to the ever-increasing volume of malware, traditional malware analysis methods, such as manual heuristic scanning, are no longer deemed dependable or efficient. That is why it is so important to implement automated, behavior-based malware detection utilizing machine learning approaches. Because malware is becoming more sophisticated at the same rate as technological advancements, the conflict between security experts and its creators will continue indefinitely. Due to its capacity to keep up with malware progress, machine learning approaches are now the focus of innovative malware detection research.

Unlike prior approaches, this one can unearth even well-hidden virus. While malware detection has previously relied on a few classic machine learning techniques, research found that XGBoost learning provided far more accurate findings.

3.2 Data Acquisition:

The first step is to collect some data for use in training algorithms, which is necessary before the research can get down to work. The data set comprises of malware data set and benign instance data set. The malicious and benign samples are both stored as Windows Portable Executable (PE) files. There are a total of 10540 samples, 6999 of which are dangerous (the "malicious data ") and 3450 of which are benign (the benign data"). Information such as Name, MD5, Hardware, SizeOfOptionalHeader, Features, Major/Minor Linker Version, Code Size, and Line Length etc.

3.3 Preparation of the data set

Initially, Executable files containing malware were acquired from "https:// virusshare.com" and used to compile the dataset. Since there were not too many clean executables available, the Dataset was built from a large collection of files found inside Windows, particularly.dll files. The dataset was constructed from the JSON output files that were uploaded to the cuckoo sandbox and thereafter subjected to a real-time analysis.

Data sets to be utilized and documented are selected in this step. Malware analysis from cuckoo sandbox reports is stored in JSON files, and for machine learning models to work on these data variables, there is a need to convert the variables to an integer readable format before extracting the variables in a CSV. When it comes to data quality, missing numbers are a big cause for alarm. Mean, mode, and median may be used instead of some of the other approaches, depending on the qualities in question. If there's a significant chance of data loss, then deleting rows and columns is a must.

There are a total of 10540 files in our combined collection, 6999 of which are dangerous and 3540 of which are safe. Since all of the files were already in. json format, no further work was necessary to transform them into.csv format for usage with the python and pandas

package. The file contained several columns, including Size, size of code, DLL characteristics, MD5 values, SHA-a values, etc., but there is only need for a subset of those columns—DLL characteristics, Entropy, ImageBase, Size of initialized data, Size of uninitialized data, and Size—because they describe the most crucial aspects of malware's potential impact on a system.

It's important to format data correctly and to re-format it if required. It is possible to execute mathematical operations on string values that hold numbers by converting them to numeric values.

4. DESIGN SPECIFICATION

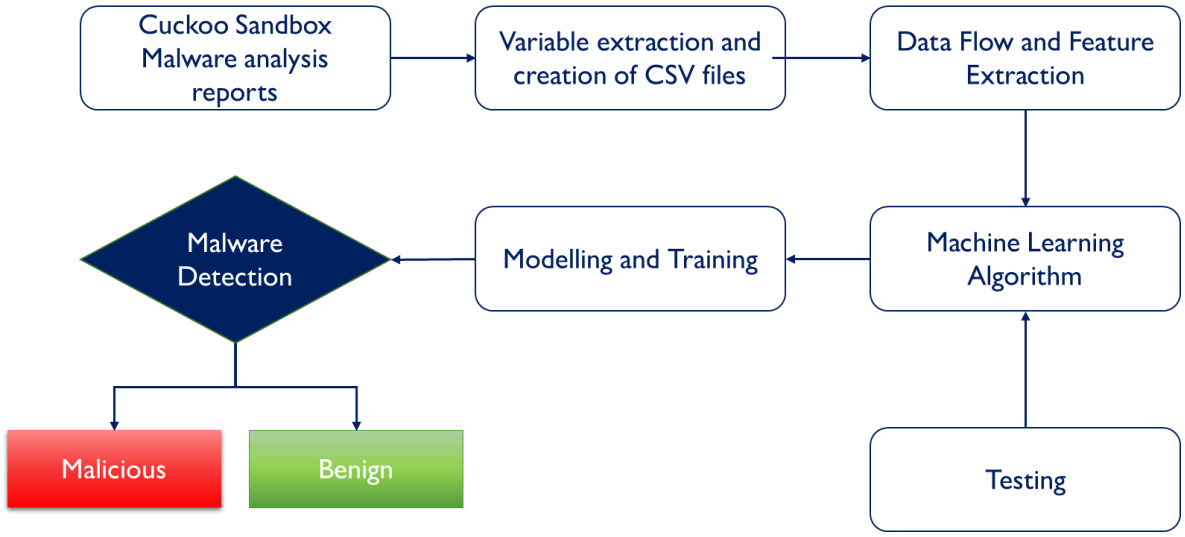

Figure 2 Implementation Design

One way to measure how well a classifier or classification algorithm is doing its job is with the use of a table called a confusion matrix. Building such a database allows one to keep track of the number of properly and erroneously labeled examples, which can subsequently be utilized in future. Accuracy, precision, and recall are the primary indicators of a categorization model's effectiveness.

Each machine learning algorithm/model was then updated to make use of our newly acquired dataset. Depending on several characteristics, the call may be classified as Malware or Goodware and return one of two possible outcomes. The dataset is used to deploy several machine learning classification algorithms and then compare how well each one handles the high dimensionality of the dataset. Nonetheless, with the help of data pre-processing, picking the right machine learning model may boost model efficiency.

5. IMPLEMENTATION

5.1 KNN Model

The KNN model comes in handy since it sorts information by using the most similar information. Its purpose is to discover all the nearest neighbors surrounding a new unknown data point to figure out what class it belongs to. Specifically, it is a distance-based method.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

Figure 3 KNN Machine Learning code

5.2 Random Forest

Each tree in a random forest uses a subset of features to make classifications and provide outputs. Several different approaches were tried before settling on 50 estimates as the optimal number for this task, with the second parameter being the Function for evaluating the division's quality [12]. Using a random sampling method, random forest constructs a decision tree and then takes an average of the results.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 50, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Figure 4 Random Forest Classifier Code

5.3 XGBoost

As part of the Distributed Machine Learning Community's bigger set of open-source libraries, XGBoost was created by Tianqi Chen (DMLC). The XGBoost algorithm is a fast and efficient implementation of gradient boosted decision trees. The maximum depth (max depth), the learning rate (learning rate), and the number of estimations is the Algorithm's parameters (n estimators). Due to its focus on model performance and computational speed, XGBoost is a scalable and accurate implementation of gradient boosting machines. It has been shown to push the boundaries of processing power for boosted tree methods.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Fitting xgboost to the training Set
from xgboost import XGBClassifier
classifier = XGBClassifier(max_depth=20, learning_rate=0.3, n_estimators=150)
classifier.fit(X_train, y_train)

#predict the test results
y_pred = classifier.predict(X_test)
```

Figure 5 XGBoost Algorithm Code

6. EVALUATION

The purpose of this study was to statistically evaluate how well existing binary categorization (malign or benign) tests work. True positive rate (sensitivity, recall, hit rate), false positive rate (fall-out), positive predictive value (precision), and accuracy are the statistical measurements [9]. Metrics are calculated in machine learning with the use of a confusion matrix included in the sklearn metrics package. It is possible to see the model's computed precision, accuracy, recall, and F score, as well as the findings provided in the model's Confusion matrix.

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figure 6 Confusion Matrix Model

- Number of samples that were accurately identified as positive (true positive, or TP).
- Number of samples that were successfully identified as negative (true negative, TN).
- Number of false-positive results (FP) indicates the percentage of negative samples that were mistakenly interpreted as positive.
- False-negative (FN) rate is the proportion of true-positive samples that were wrongly classified as negative.

The other parameters can be calculated as follows:

- The effectiveness of the categorization algorithm will also be measured by the following criteria

$$\text{Accuracy} = (TP + TN) / (TP + TN) + (FP + FN)$$

- The ratio of accurate to wrong classifications is calculated using the formula:

$$\text{Precision}=TP/(TP+FP)$$

- Number of accurate classifications minus penalty for number of missing elements

$$\text{Recall}=TP/(TP+FN)$$

- An efficient metric based on the harmonic mean of precision and recall is the F-score, which is calculated as follows:

$$\text{F1-score}=2*(\text{Precision}*\text{Recall})/ (\text{Precision} + \text{Recall})$$

This Research concludes that the XGBoost method achieves a higher rate of accuracy than competing algorithms based on the assessment and the outcomes from the models that were studied. Our algorithm can identify and categorize a wide variety of malicious and safe files.

6.1 RESULTS

The dataset was then used to train a model or algorithm, and the resulting data was collected. Accuracy, Precision, Recall, and F1-score were calculated by solving the confusion matrix for each model.

6.1.2 Results without Feature extraction

a) KNN

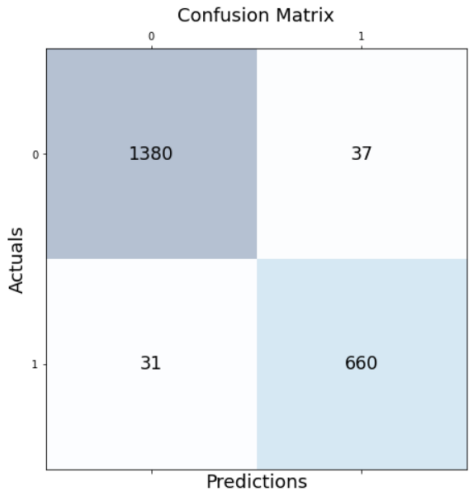

Figure 8 KNN Confusion Matrix without feature selection

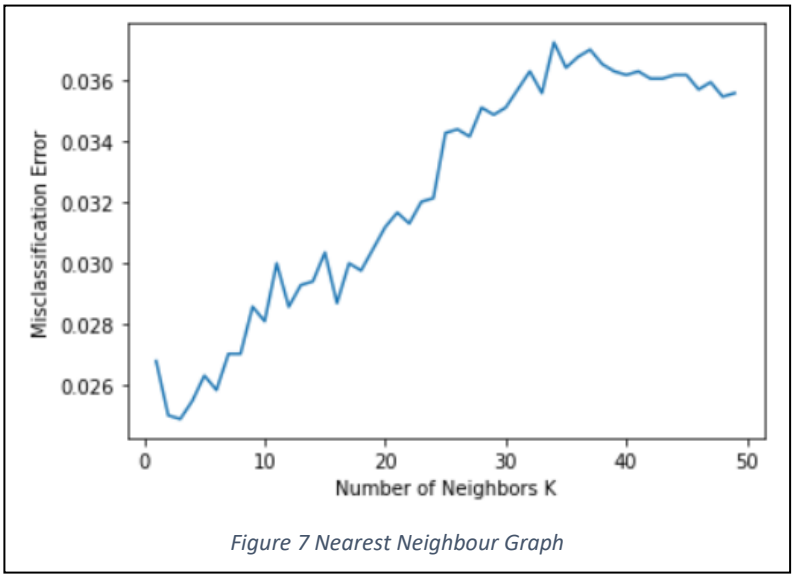

Figure 7 Nearest Neighbour Graph

We will use CrossValidation (a statistical technique for evaluating the quality of machine learning models) to determine the best value of K. The technique relies on the user providing a value for K that stands for the total number of neighbors. The method may be performed with a variety of different values for K to see which one works best for the given situation.

The optimal number of neighbors is 3. The model gives out 96.774 % accuracy with 97.388% of precision.

b) Random Forest

The confusion matrix below shows the accuracy and the precision

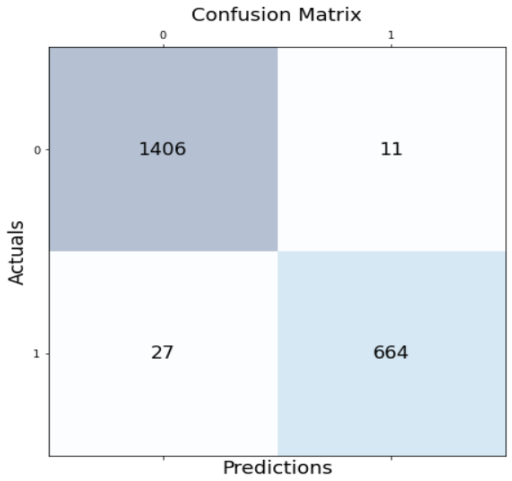

Figure 9 RF Confusion Matrix without feature selection

This model has an accuracy of 98.197% and a precision of 99.223%, which is a significant improvement over the KNN model.

c) XGBoost

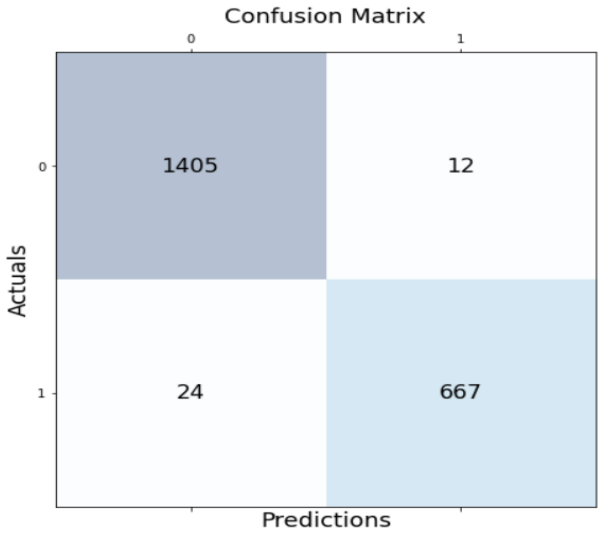

Figure 10 XGB Confusion Matrix without feature selection

The Overall accuracy and the precision for this model is the best among all the machine learning classifiers. The precision and the accuracy increased by not much though but overall, an increase is seen. The Accuracy of this model comes out to be 98.292% and the precision for this model is 99.153%.

6.1.3 Results with Feature Extraction

The purpose of feature selection is to reduce our 54-feature dataset to the most relevant features that may be utilized to identify benign from malicious files. A plan to apply Tree-based feature selection to investigate whether it enhances the accuracy and performance of the algorithms is made.

The "sklearn model" import "train test split" in Python causes a split to occur. Because it's more efficient to use just a subset of the data to training, the dataset is often divided in half, with 80% used for training and 20% used for testing. Below, a detailed explanation of each model, including the KNN model, the SVM model, the [] model, and the XGBoost model is given. The purpose of feature selection is to reduce our 54-feature dataset to just the most relevant features that may be utilized to identify benign from malicious files. To determine whether or if this strategy enhances accuracy and algorithms, Usage of Tree-based feature selection is chosen. This algorithm only picked 9 out of 54 necessary attributes.

a) KNN

For this model the optimal number of neighbors is 6. As compared to the previous model without feature extraction, the accuracy and the precision slightly increased to 97.438% and 98.023% respectively which is 0.68% increase in accuracy and 0.65% increase in precision

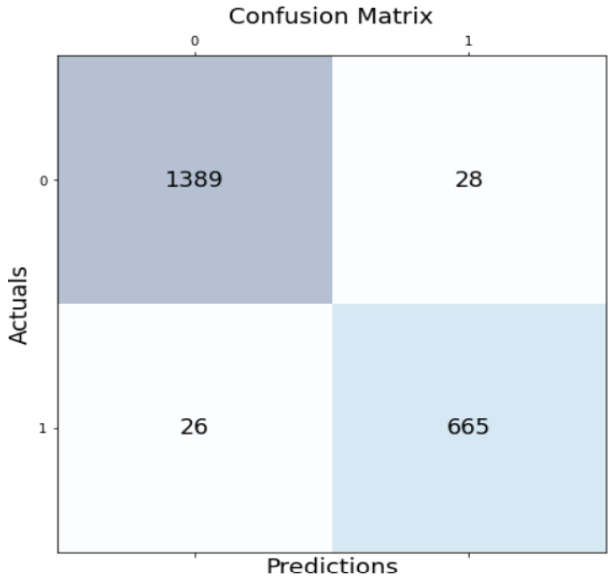

Figure 12 KNN Confusion Matrix with feature selection

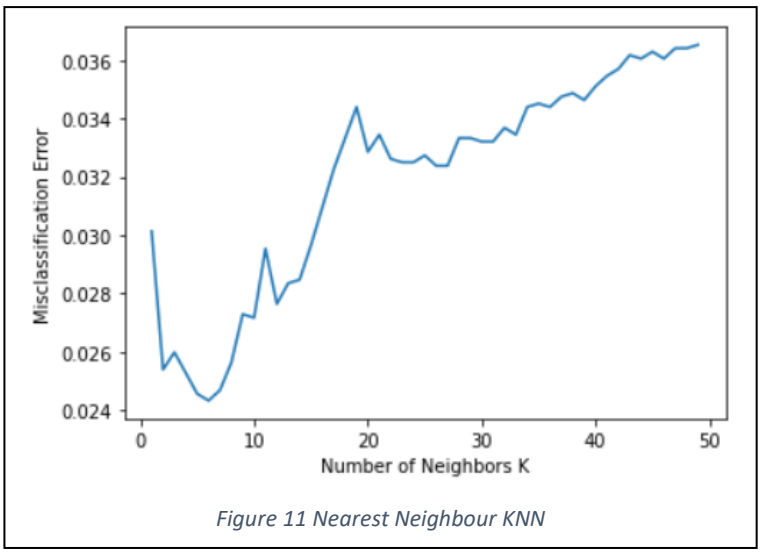

Figure 11 Nearest Neighbour KNN

b) Random Forest

After the feature extraction procedure, the random forest model's classifier is less successful, as shown by the fact that it achieves an accuracy of 97.67% and a precision of 98.08% in its classifications.

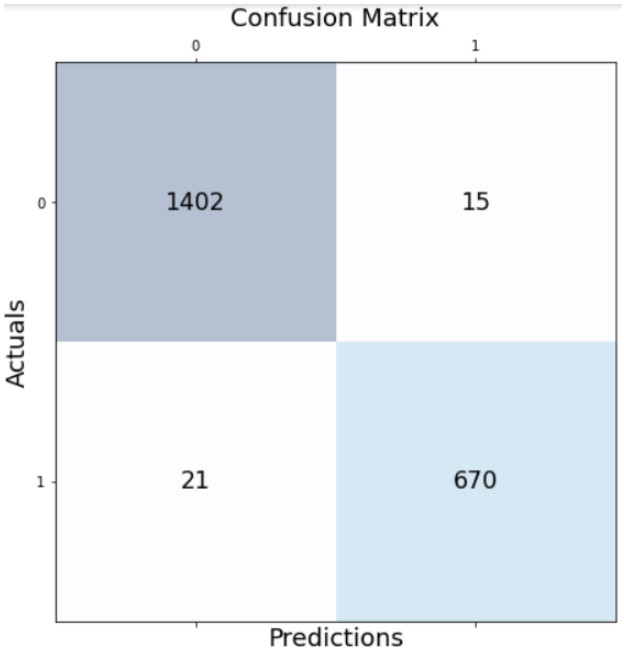

Figure 13 RF Confusion Matrix with feature selection

c) XGBoost

Among the models that make use of feature selection, the XGBoost model produces the best outcomes, with an accuracy of 97.67% and a precision of 98.23%.

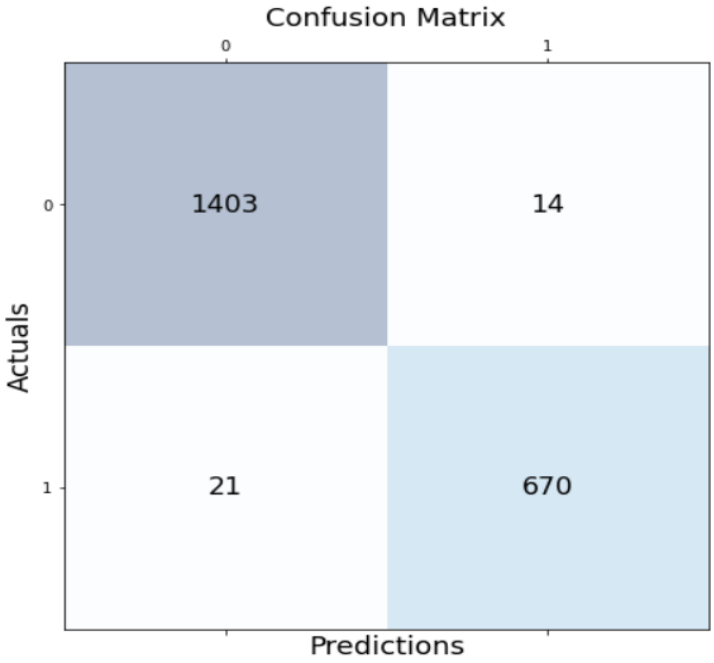

Figure 14 XGBoost Confusion Matrix with feature selection

The Accuracy, precision, recall, and F1-score are calculated using the confusion matrix for each model and the data represented in the above diagrams. There is an increase in the accuracy level of data with feature selection.

6.2 DISCUSSION

Most of the work done on malware detection have used various datasets and some have used different algorithms. In this research the focus is given on the Latest machine learning algorithms XGBoost and how its analysis the data and classifies out input to either benign or malign executables

The dataset utilized in this work is around typical in size when compared to those used by other researchers [3][6]. An accuracy of 97% or higher is achieved, which is fine since this is a dataset of medium size. Since the dataset is unbalanced, this research was unable to do thorough analyses. Consistently high levels of accuracy and precision have been found in related studies. Furthermore, the accuracy and precision in this study demonstrates that XGBoost has the upper hand amongst other classifiers, since previous studies using the same features and other datasets have only achieved an accuracy of 95% with SVM [2][3], while the accuracy with XGBoost is nearly 98.3% in all situations.

Although after the feature selection is implemented, the values of the precision and accuracy increased in the KNN model but decreased by 0.63% in both Random Forest and XGBoost cases. The accuracy and the precision remain almost same for XGBoost with and without feature selection as there is a very slight increase in accuracy of model with feature selection.

7. CONCLUSION AND FUTURE WORK

This research shows how the malicious executables can be found out in the wild with the help of machine learning. In the initial part of the process, data of both malicious and benign executables are employed to build our data collection, and a Python script is used to extract the information needed. It is necessary to get the data set ready for training machine learning algorithms once it has been created. After implementation of the model, the model got an accuracy of 98.29% in the XGB model followed by 98.19% of Random Forest, and second-highest accuracy by KNN which is 96.77%. The research expected that the XGBoost will give more accuracy and it do so with an increase of 0.065 % from Random Forest algorithm and 1.54 % increase from KNN algorithm in accuracy. The Recall and F1 scores are the best achieved by XGBoost with feature selection where recall is 98.52% an F1 score is coming out to be 98.76% Overall, the best results are given with feature extraction and the best accuracy is given by XGBoost algorithm.

Table 1 Summarised Results

Classifiers	Accuracy	Precision	Recall	F1 Score
KNN	0.967741935	0.973888497	0.978029766	0.975954738
RF	0.981973435	0.992237121	0.981158409	0.986666667
XGB	0.982922201	0.991531404	0.983205038	0.987350668
KNN*	0.973908918	0.979534227	0.981612447	0.980572236
RF*	0.982922201	0.989414255	0.985242446	0.987323944
XGB*	0.983396584	0.990119972	0.985252809	0.987680394

*= Models with Feature Selection

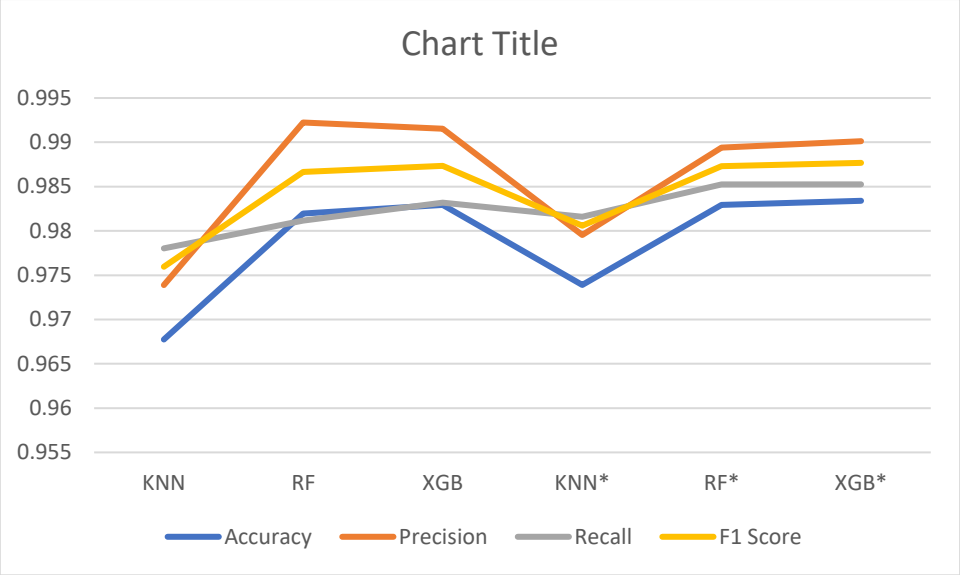

Figure 155 Graphical Representation of Summarised Results

Although when the features were selected in XGBoost, the accuracy increased a bit for (0.043%), but the precision decreased by 0.142%. If after feature selection accuracy or precision falls, it means that all the variables are important for the model when we are minimising the false positives. As the precison of the algorithm XBoost The Graphical representation shows the efficacy of all the models with respect to each other

7.1 LIMITATIONS:

Due to time constraints, the study was unable to demonstrate the behavior of executables by extracting the API call sequence for each file, analysing, or seeing the behavior of the files through the numerous API calls made by each file, and conducting detection appropriately. Despite the study's success in improving accuracy, it would have been much more effective if it had been used to a much larger dataset, where file behavior may have yielded more precise findings. Although malicious and safe files were identified thanks to this study, the effectiveness of the model classification might be improved with the help of behavior statistics.

7.2 FUTURE WORK

The future work can classify files in the future based on the actions of executables by extracting APIStats variables from Cuckoo Sandbox data and improving the model [11]. They may train machine learning models based on the API calls of the files, evaluating the behavior of malware kinds, and attempting to identify the malware in real time if they compile the API calls of malicious and benign files into a phrase or frequency dictionary. By compiling the dataset, there may be a possibility of getting higher accuracy and precision results.

8. ACKNOWLEDGMENT

Finally, I'd want to express my gratitude to my advisor, Prof. Imran Khan, who was there for me during the whole process of writing this report and who helped me make informed decisions throughout. In addition, I want to express my gratitude to my parents for their unwavering love and support I encountered while working on my thesis. For his assistance with the implementation whenever I hit a snag, I'd want to extend my gratitude to a buddy who is a student in data analytics at NCI. As an added note, I'd like to express my appreciation to the National College of Ireland and the School of Computing for successfully wrapping up their respective semesters.

9. REFERENCES:

[1] Catak, F.O. and Yazı, A.F. (2021). A Benchmark API Call Dataset for Windows PE Malware Classification. *arXiv:1905.01999 [cs]*. [online] Available at: https://arxiv.org/abs/1905.01999

[2] Firdausi, I., Erwin, A., Lim, C. and Satriyo Nugroho, A. (2010). *(PDF) Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/232627329_Analysis_of_Machine_learning_Techniques_Used_in_Behavior-Based_Malware_Detection [Accessed 11 Nov. 2022].

[3] Galal, H.S., Mahdy, Y.B. and Atiea, M.A. (2015). Behavior-based features model for malware detection. *Journal of Computer Virology and Hacking Techniques*, 12(2), pp.59–67. doi:10.1007/s11416-015-0244-0.

[4] Hotz, N. (2022). *CRISP-DM*. [online] Data Science Project Management. Available at: https://www.datascience-pm.com/crisp-dm-2/ [Accessed 25 Nov. 2022].

[5] Hussain, A., Asif, M., Ahmad, M.B., Mahmood, T. and Raza, M.A. (2022). Malware Detection Using Machine Learning Algorithms for Windows Platform. *Lecture Notes in Networks and Systems*, pp 619–632(https://doi.org/10.1007/978-981-16-7618-5_53), pp.619–632. doi:10.1007/978-981-16-7618-5_53.

[6] Kolter, J.Z. and Maloof, M.A. (2004). Learning to detect malicious executables in the wild. *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04*, 2721-2744. doi:10.1145/1014052.1014105.

[7] Naz, S. and Singh, D.K. (2019). *Review of Machine Learning Methods for Windows*

Malware Detection. [online] IEEE Xplore. doi:10.1109/ICCCNT45670.2019.8944796.

[8] Radwan, A.M. (2019). *Machine Learning Techniques to Detect Maliciousness of Portable Executable Files*. [online] IEEE Xplore. doi:10.1109/ICPET.2019.00023.

[9] Sarang Narkhede (2018). *Understanding Confusion Matrix*. [online] Medium. Available at: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 [Accessed 23 Nov. 2022].

[9] Schultz, M.G. and Eskin, E. (n.d.). *Data Mining Methods for Detection of New Malicious Executables*. [online] Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a9bd1102fbd9a3184127a77a50dc47517147a3ba [Accessed 11 Nov. 2022].

[10] Shiva Darshan, S.L. and Jaidhar, C.D. (2018). Windows malware detection system based on LSVC recommended hybrid features. *Journal of Computer Virology and Hacking Techniques*, 15(2), pp.127–146. doi:10.1007/s11416-018-0327-9.

[11] Singh, J. and Singh, J. (2020). Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Information and Software Technology*, 121, p.106273. doi:10.1016/j.infsof.2020.106273.

[12] Ujhelyi, T. (2022). *Random Forest in Python and coding it with Scikit-learn (tutorial)*. [online] Data36. Available at: https://data36.com/random-forest-in-python/ [Accessed 26 Nov. 2022].

[13] Wang, C., Ding, J., Guo, T. and Cui, B. (2017). A Malware Detection Method Based on Sandbox, Binary Instrumentation and Multidimensional Feature Extraction. *Advances on Broad-Band Wireless Computing, Communication and Applications*, pp.427–438. doi:10.1007/978-3-319-69811-3_39.