

Detection & Classification of Obfuscated Malware

20BCE1602 - Ashutosh Pathak

Table of Contents

[Abstract 2](#_Toc22869)

[Introduction 3](#_Toc22870)

[MOTIVATION: - 3](#_Toc22871)

[PROBLEM STATEMENT: - 3](#_Toc22872)

[OBJECTIVES: - 3](#_Toc22873)

[METHODOLOGY: - 3](#_Toc22874)

[Literature Survey 3](#_Toc22875)

[Random Forest for Malware Classification 3](#_Toc22876)

[Early-Stage Malware Prediction Using Recurrent Neural Networks 4](#_Toc22877)

[Machine Learning-Based Intrusion Detection System 4](#_Toc22878)

[Machine Learning Classification Model For Network-Based Intrusion Detection System 5](#_Toc22879)

[Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time 5](#_Toc22880)

[Deep Neural Network-Based Malware Detection Using Two-Dimensional Binary Program Features 6](#_Toc22881)

[Applying NLP techniques to malware detection in a practical environment 6](#_Toc22882)

[Proposed Work 7](#_Toc22883)

[DATA SOURCE: - 7](#_Toc22884)

[DATA ANALYTICS MODEL: - 9](#_Toc22885)

[ALGORITHM DESCRIPTION: - 9](#_Toc22886)

[Results & Discussion 15](#_Toc22887)

[References 20](#_Toc22888)

# Abstract

Detecting and blocking malware has become increasingly challenging for antivirus programs and other security measures due to the constantly changing sophisticated obfuscation techniques. This further leads to increased risks for users. Obfuscation is used to hide malicious code within legitimate processes, making it difficult to detect.

In this project, we propose a machine learning-based approach that uses system call sequences as features to detect different types of malware, such as Ransomware, Trojan, Spyware, and benign (or genuine) processes. We evaluated our model on the dataset of CIS-MalMem2022 and achieved an overall accuracy of ***87.09%***. The results demonstrate the effectiveness of our approach in detecting different types of malware and benign processes.

Keywords – Malware, Obfuscation, CTGAN, SMOTE, Hyperparameter Tuning

# Introduction

## MOTIVATION: -

The motivation for this project stems from the fact that obfuscated malware has become increasingly prevalent. It is a growing concern in the field of cybersecurity. Attackers continuously find new ways to obfuscate their malicious code to evade detection and achieve their nefarious goals. As a result, this field requires greater attention to develop new techniques and strategies to detect and mitigate obfuscated malware.

## PROBLEM STATEMENT: -

The problem statement identified is to detect the type of malware and classify them as either “trojan,” “spyware,” or “ransomware,” or even “benign,” based on their various attribute values.

## OBJECTIVES: -

Our objective is to identify and classify obfuscated malware effectively.

## METHODOLOGY: -

To achieve our target, we have used supervised and unsupervised learning models to train and test on our dataset at different resource levels. We have also used hyperparameter tuning on the different attributes to identify and neglect the outliers in the dataset.

Implementing all these techniques, we have identified Random Forest as the most suitable model for identifying and classifying the type of malware.

# Literature Survey

### **Random Forest for Malware Classification**

**Introduction:** Random Forest is a machine-learning algorithm that can be used for malware classification. It works by constructing multiple decision trees and combining their predictions to classify a new sample. The algorithm is known for its robustness, accuracy, and ability to handle imbalanced datasets.

**Methodology:** To use Random Forest for malware classification, one must create a dataset of features extracted from malware samples and use the algorithm to train a model on this data. The trained model can then classify new malware samples into different categories, such as benign or malicious.

Its methodology includes Data Preparation, Classification, Training, and Validation. The dataset used is the Malimg Dataset consisting of 9,342 malware samples of 25 different malware families.

**Result:** The resulted confusion matrix displays the accuracy of classifying each malware family and reveals a similar trend, particularly for the four malware families (CL2OP.gen!g, C2LOP.P, Swizzor.gen!E, Swizzor.gen!I) with an accuracy below 0.5. Additionally, the data illustrates that there are misclassifications between the CL2OP.gen!g, C2LOP.P, Swizzor.gen!E, and Swizzor.gen!I malware families.

### **Early-Stage Malware Prediction Using Recurrent Neural Networks**

**Introduction:** Early-stage malware prediction using recurrent neural networks is a topic of interest in the field of cybersecurity. This paper mainly focuses on using recurrent neural networks (RNNs) to detect malware in its early stages before it can cause damage to a system.

**Methodology:** The RNNs have shown promising results in early-stage malware detection, as they can capture the temporal relationships between different system events and detect patterns that indicate the presence of malware. Some studies have used dynamic RNNs, which can adapt to changing malware behavior, to improve the detection of evolving malware threats. To address the imbalance between the number of benign and malicious samples in the training data, some studies have employed techniques such as oversampling and undersampling to balance the dataset. The feature selection process is crucial for the performance of RNNs in malware detection, as it determines the relevant information fed into the model.

**Result:** RNNs can be effective in early-stage malware detection. Their performance can be improved by combining them with other machinelearning algorithms and carefully selecting the features used as input.

### **Machine Learning-Based Intrusion Detection System**

**Introduction:** Due to the large volume of data, the number of false alarm reports of network intrusion increases, and detection effectiveness decreases. This is one of the main problems when an unidentified attack hits the system. The primary goals are to lower the false alarm rate and raise the accuracy rate. Machine learning algorithms like SVM and Naive Bayes have addressed the abovementioned difficulties.

**Methodology:** A comparative analysis was done between SVM and Naïve Bayes to classify the dataset and analyze their accuracy.

Support Vector Machine-> generates one or more hyperplanes in a highdimensional space. The best hyperplane divides the given data into different classes with the primary partition as efficiently as possible.

Naive Bayes -> Statistical classifiers include Bayesian classifiers. They can predict the likelihood that a given model would fit a particular class. Its foundation is the Bayes theorem. The suggested IDS can distinguish between known and unknown assaults.

**Result:** For 19,000 instances, the accuracy rate has been analyzed to compare the effectiveness of SVM and Naive Bayes.

Accuracy: -

SVM -> 97.29

Naïve Bayes -> 67.26

SVM-CfsSubsetEval -> 93.95

Naïve Bayes-CfsSubsetEval -> 56.54

### **Machine Learning Classification Model For Network-Based Intrusion Detection System**

**Introduction:** This research proposes a Machine Learning (ML) based model for Network-based Intrusion Detection Systems (NID) to detect advanced mobile threats such as malware that can lead to stealing sensitive information, installing backdoors, ransomware attacks, and sending premium SMSs. The ML model uses supervised classifiers built from labeled instances of network traffic features generated by malicious and benign applications. The model was evaluated on Android-based malware due to its popularity and high global share in mobile malware.

**Methodology:** Several datasets are commonly used in network-based intrusion detection (NID) systems, including KDD99, DARPA 1998/1999, and ISCX 2012 IDS dataset. However, these datasets do not apply to Android-based attacks, and there is a need for a reference labeled dataset to evaluate the performance of NID for Android traffic. There are also several datasets of mobile malware samples, including the MalGenome, Drebin, and ISCX Android Botnet datasets. These limitations highlight the need for a more robust solution for Android malware detection.

**Result:** This research evaluated the performance of machine learning (ML) classifiers for detecting malicious traffic. The experiment showed that the TPR (True Positive Rate) for the classifiers was between 99.0-99.6% for the training set and 93.8-100% for the unseen dataset, while the FPR (False Positive Rate) was between 1.8-6.5% for the training set and 0-11.7% for the unseen dataset. The study concluded that ML classifiers are more efficient than traditional antivirus software in detecting malicious traffic. The model built in this study had an accuracy of 97.5% on unseen data.

### **Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time**

**Introduction:** Detection of malware using deep learning techniques like Convolutional Neural Networks (CNN) and Long-Short-Term Memory (LSTM) is a promising approach for real-time malware detection.

**Methodology:** This method trains a deep learning model on a benign and malicious software dataset and uses the model to classify new unknown software as benign or malicious. The CNN component of the model extracts features from the software's behavior. In contrast, the LSTM component remembers the sequential relationships between the features over time, allowing for a more sophisticated analysis of the software's behavior. This approach aims to accurately identify malware in real-time, even if the malware is disguised or mutated.

**Result:** Malware classification technique using the proposed methodology. An effective classification rate with an accuracy of 100%, 91%, and 66.67% using J48, SMO, and Random Forest Tree, respectively, has been observed in the experiment. It is reported that the decision tree yields an accuracy of 100% because of the small sample size.

### **Deep Neural Network-Based Malware Detection Using Two-Dimensional Binary Program Features**

**Introduction:** The paper mentioned above introduces a deep neural network-based malware detection system that Invincea Labs has developed. It achieves a usable detection rate at a meager positive false rate and then

scales to real-world training example volumes on commodity hardware. Thus, the model promises an extremely high accuracy rate with less training data.

**Methodology:** For most algorithms training the model is complex due to a lack of training data, but this approach solves the lack of it. The system achieves a **95%** detection rate at a **0.1%** false positive rate, based on more than **400,000** software binaries sourced directly from consumers and internal malware databases.

In addition, a non-parametric method is described for adjusting the classifier’s scores to better represent expected precision in the deployment environment.

**Results:** The results show that it is now feasible to quickly train and deploy a low-resource, highly accurate machine-learning classification model with false positives rates that approach traditional labor-intensive expert rulebased malware detection while also detecting previously unseen malware missed by these conventional approaches.

Since machine learning models tend to improve with more extensive data sizes, deep neural network classification models are gaining importance as part of a layered network defense strategy in the coming years. The feature engineering includes Byte/Histogram features, PE import features, String 2D histogram, and PE Metadata. Based upon those features are aggregated for labeling.

### **Applying NLP techniques to malware detection in a practical environment**

**Introduction:**Targeted attacks are one of the most severe threats through the Internet. Attackers often use obfuscated malware to evade anti-virus programs. The idea is to force any suspicious binary to execute in sandboxes, and if their behaviors are malicious, then the file is classified as malware. With the recent development of natural language processing (NLP) techniques, printable strings have become more effective in detecting malware. In this paper, we apply NLP techniques to malware detection.

**Methodology:**

***NLP-based detection*** *-* Our detection model uses some NLP techniques. This section focuses on NLP-based detection methods.

***Bag-of-words -*** is a fundamental method of document classification where the frequency of each word is used as a feature for training a classifier.

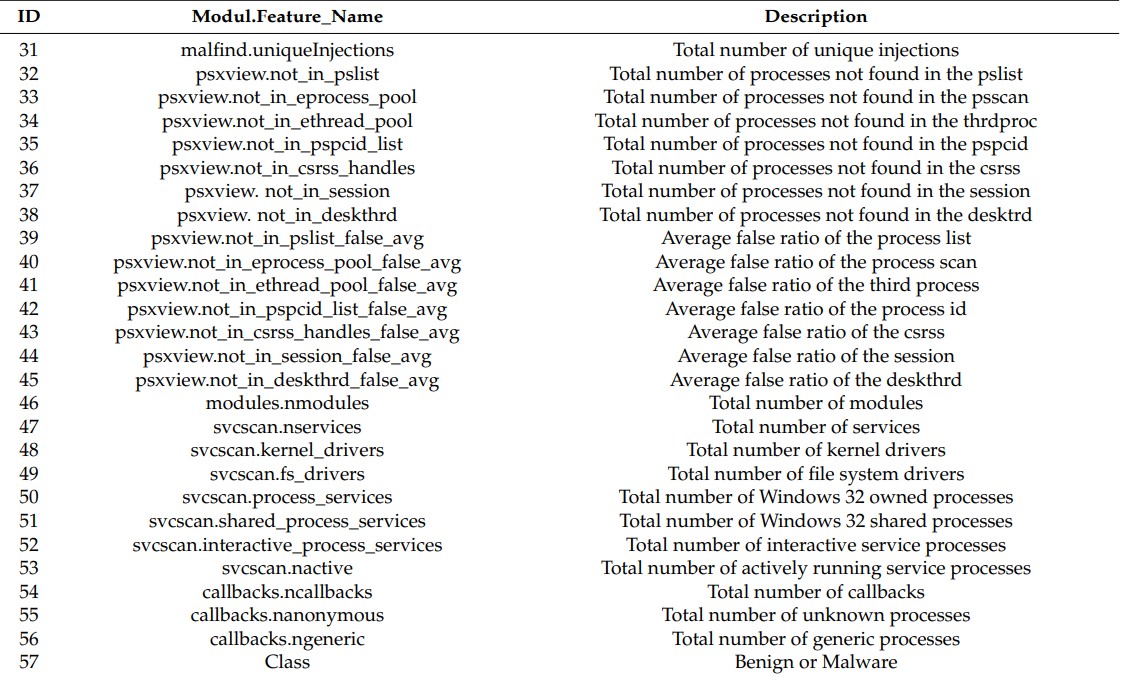
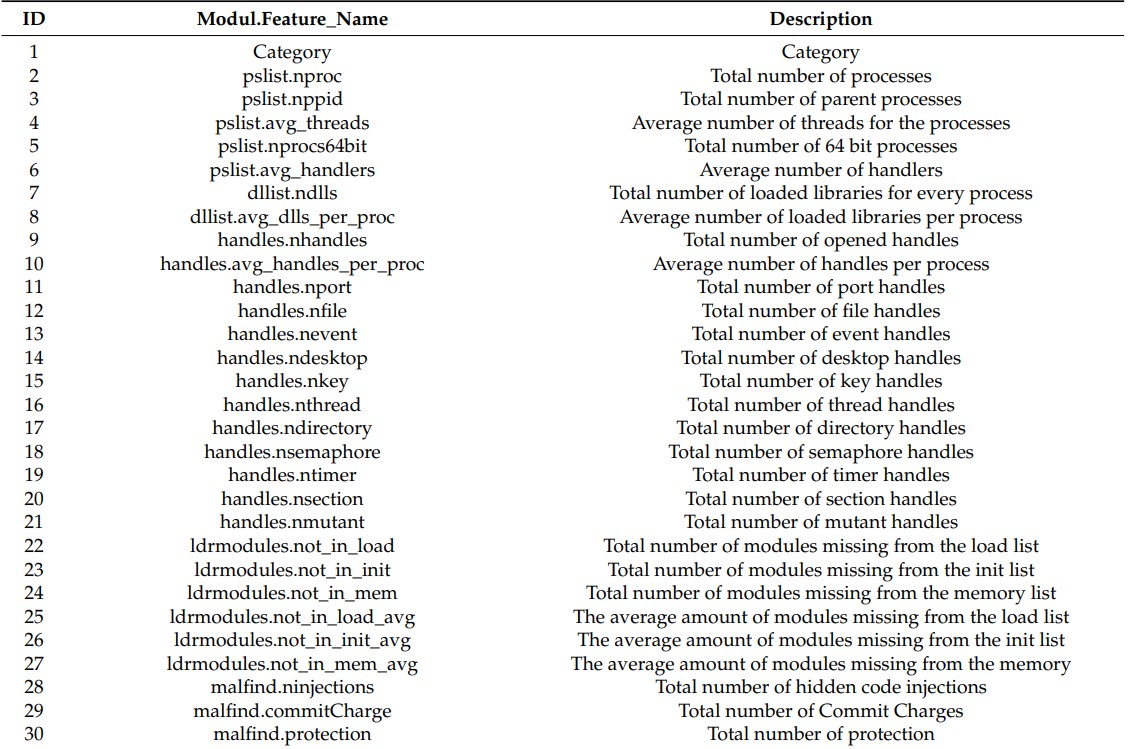
***Latent semantic indexing*** *-* LSI analyses the relevance between a document group and words included in a document.

**Result:** Printable strings with NLP techniques effectively detect malware in a realistic environment. Our dataset consists of more than 500,000 samples obtained from multiple sources. Our detection model is effective against packed malware and anti-debugging techniques. Despite this, our study could be a starting point to evaluate practical performance.

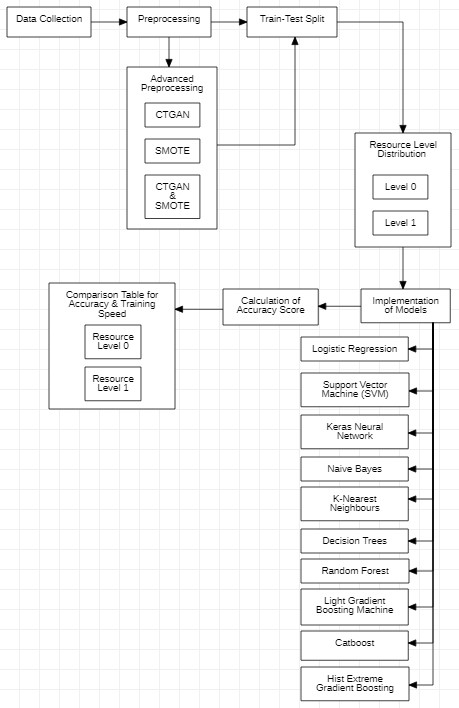
# Proposed Work

## DATA SOURCE: -

We are using obfuscated malware dataset from the [site.](https://www.unb.ca/cic/datasets/malmem-2022.html) The obfuscated malware dataset is designed to test obfuscated malware detection methods through memory. The dataset was created to represent as close to a realworld situation as possible using malware that is prevalent in the real world. Made up of Spyware, Ransomware, and Trojan Horse malware, it provides a balanced dataset that can be used to test obfuscated malware detection systems. The dataset has 57 attributes based on which we are classifying the category. The feature description is as follows:



## DATA ANALYTICS MODEL: -



For the model, we have used two different types of parameters. First, with resource level 0, which uses a basic hyperparameter, for resource level 1, we have improved the hyperparameters.

## ALGORITHM DESCRIPTION: -

We had initially applied 10 machine learning algorithms, which are mentioned below. To further improve upon the learning of the applied models and the obtained accuracy, we used CTGAN to generate more data and SMOTE to treat data imbalance.

Finally, we have applied a combination of both CTGAN and SMOTE on our dataset and applied those 10 models again to obtain an accuracy of 94.27%.

1. **Pre-processing:** The *drop()* function from the panda’s library first removes the 'Class' and 'Category' columns from the input data. The 'Class' column is unnecessary for modeling purposes, while the 'Category' column works as our target variable.

To predict the category of a sample, we used the *replace()* function from the panda’s library to replace the categorical labels with numerical labels. 'Benign' was replaced with 0, 'Ransomware' with 1, 'Spyware' with 2, and 'Trojan' with 3. Thus, the categorical labels in the 'Category' column are transformed into numerical labels for the following machine-learning models.

1. **Splitting the Dataset:** To test whether the predicted classes are correct or not, we had to try the trained model. So, we split the existing dataset into train and test based on 70-30% criteria.
2. **Resources Level:** Resources level and number features are the two arguments that the model functions we built takes in as formal parameters, providing the hyperparameters for the models. The hyper param dictionary, which is initialized to an empty dictionary, contains the hyperparameter definitions.

If resources\_level is 0, the function sets the hyperparameters for low resource models. These models are made to be computationally less intensive and use fewer resources.

While, if resources\_level is 1, the function sets the hyperparameters for high resource models, which makes them more computationally intensive and utilizes more resources.

1. **Logistic Regression**: Logistic regression is a statistical method used to analyze and model the relationship between a binary dependent variable and one or more independent variables. The classifier uses the Scikit-learn library for machine learning. It is used to build a model for a classification task. The solver argument is set to 'sag,’ which stands for Stochastic Average Gradient descent. Large-scale logistic regression issues are solved with this solution since it is computationally effective and capable of handling numerous data. The max\_iter argument is set to hyper-param["lr\_max\_iter"], which is the maximum number of iterations for the solver to converge. This value is defined in the hyper param dictionary, which is passed as an argument to the built function.

The initialized classifier is kept in the dictionary called built models under the “Logistic Regression” key and saved.

The accuracy obtained at resource level 0 is **52.36%,** and at resource level 1 is **52.05%.**

Merit: Logistic regression is simple to implement and interpret. Demerit: Logistic regression assumes a linear relationship between predictors and the log odds of the outcome.

1. **Support Vector Machine:** Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression analysis. This classifier is used to build a model for a classification task. When the kernel option is set to "linear," the SVM will operate using a linear kernel. For data that can be separated linearly, this kernel fits well. The random\_state argument is set to cur\_random\_state to produce a random number of seeds used to ensure the consistency of outcomes. The max\_iter argument is set to hyperparam["svc\_max\_iter"], which is the maximum number of iterations for the solver to converge.

The built\_models dictionary is used to store the initialized classifier. The classifier is stored under the “Support Vector Machine” key in this case.

The accuracy obtained at resource level 0 is **65.82%,** and at resource level 1 is **48.66%.**

Merit: SVM can handle high dimensional data and effectively deals with non-linearly separable data.

Demerit: SVM can be sensitive to the choice of kernel function and parameters.

1. **Keras Neural Network:** Keras is an open-source neural network library written in Python designed to enable fast experimentation with deep neural networks. This model is used to build a neural network for the classification task.

The model consists of four layers. The first layer is a Flatten layer, which is used to flatten the input data into a one-dimensional array. The second layer is a Dense layer with 67 units and a ReLU activation function. The third layer is a Dropout layer with a rate of 0.2, which randomly drops out 20% of the units in the previous layer during training. The fourth layer is a Dense layer with 15 units and a softmax activation function, which outputs a probability distribution over the 15 classes.

1. The stochastic gradient descent algorithm known as the Adam optimizer is used to create the model. For multiclass classification problems with integer labels, the loss function is set to sparse\_categorical\_crossentropy.

The built\_models dictionary is used to store the initialized model. In this case, the model is stored under the key "Keras Neural Network.” The accuracy obtained at resource level 0 is **67.69%,** and at resource level 1 is **68.11%.**

Merit: Keras allows for quick and easy prototyping of deep learning models with minimal coding.

Demerit: Keras can be limited in flexibility and may not be suitable for more complex or customized neural network architectures.

1. **Naive Bayes:** Naive Bayes is a probabilistic algorithm used in machine learning for classification and prediction tasks based on Bayes' theorem. In Naive Bayes, we use the GaussianNB function for our project. This classifier is used to build a model for a classification task. The Gaussian Naive Bayes classifier assumes that the features are normally distributed and are conditionally independent given the class label.

The nb\_classifier object is used to store the initialized classifier. In this case, the classifier is stored under the “Naïve Bayes” key.”

The accuracy obtained at resource level 0 is **68.57%,** and at resource level 1 is **68.50%.**

Merit: Naive Bayes is computationally efficient and can work well with small datasets.

Demerit: Naive Bayes assumes that all features are independent of each other, which can lead to decreased accuracy in some cases.

1. **K Nearest Neighbors:** It is a non-parametric, supervised learning classifier that uses proximity to make classifications or predictions about the grouping of an individual data point. By identifying the k samples in the training set that are the most similar to the sample under consideration, the KNN classifier predicts the class of the sample by utilizing the predicted class shared by most of these samples.

The initialized classifier is stored in the KNN classifier object. The number of neighbours to consider when making predictions is indicated by the value given in the hyper-param dictionary in the n neighbors hyperparameter.

In this case, the classifier is stored under the key "K Nearest Neighbors.”

The accuracy obtained at resource level 0 is **80.91%,** and at resource level 1 is **79.88%.**

Merit: KNN is simple to implement and can work well with nonlinear data and complex decision boundaries.

Demerit: KNN can be computationally expensive for large datasets and may need to perform better with high-dimensional data.

1. **Decision Trees:** Decision tree is a supervised machine learning algorithm used for classification and regression analysis that partitions the data into a hierarchical tree-like structure based on the features of the data.

The initialized classifier is kept in the decision\_tree\_classifier object. The hyperparameter max depth is set to the maximum depth of the tree, which is the number listed in the hyper-param dictionary. The classifier, in this instance, is saved with the key "Decision Trees.” The accuracy obtained at resource level 0 is **81.70%,** and at resource level 1 is **84.33%.**

Merit: Decision trees are easy to understand and interpret and can handle both categorical and numerical data.

Demerit: Decision trees can be prone to overfitting and may need to perform better with complex data structures.

1. **Random Forest:** Random Forest is an ensemble learning technique in machine learning that builds multiple decision trees and combines their output to improve the accuracy and robustness of the model. The initialized classifier is kept in an object called

random\_forest\_classifier. The hyperparameter max\_depth is set to the maximum depth of the trees, which is the value listed in the hyperparam dictionary.

The classifier is kept in this instance under the key "Random Forest.” The accuracy obtained at resource level 0 is **82.21%,** and at resource level 1 is **87.09%.**

Merit: Random Forest can handle high-dimensional data with many features and can handle missing values.

Demerit: Random forests can be slow to train on large datasets and may need to be more easily interpretable than individual decision trees.

l) **Light Gradient Boosting Machine:** Light Gradient Boosting Machine (LightGBM) is a gradient boosting framework that uses tree-based learning algorithms and is designed to be efficient in training and prediction speed, memory usage, and scalability.

The algorithm is an ensemble learning method that utilizes trees to optimize a differentiable loss function by gradually adding weak learners to the model. This is achieved through a process of iterative improvement, where the model is trained to minimize the loss function by analyzing and improving the performance of its individual components.

The initialized classifier is kept in an object called lgbm\_classifier. The n\_estimators hyperparameter in LightGBM specifies the number of boosting iterations (i.e., the number of trees) to perform during training. Increasing the number of trees will improve the model's performance but also increase the risk of overfitting and slow down the training process.

The accuracy obtained at resource level 0 is **86.14%,** and at resource level 1 is **86.77%.**

Merit: LightGBM is highly efficient and can handle large datasets with high dimensionality, making it well-suited for big data applications. Demerit: LightGBM can be sensitive to overfitting and may require careful tuning of hyperparameters to achieve optimal performance.

m) **Catboost:** CatBoost is an open-source gradient boosting framework that uses decision trees and is designed to work well with categorical features and handle missing values.

What sets CatBoost apart is its optimized and efficient handling of categorical features, which are variables with discrete values like text labels or identifiers, during the training process.

The initialized classifier is kept in an object called catboost\_classifier. It has hyperparameter n\_estimator, which refers to the count of trees in the ensemble or the number of boosting iterations to be executed during training. It can be considered equivalent to the "number of trees" or the "number of boosting rounds" in other gradient boosting algorithms.

The accuracy obtained at resource level 0 is **86.14%,** and at resource level 1 is **86.77%.**

Merit: CatBoost can handle categorical features and missing values without requiring preprocessing and can achieve high accuracy with small and unbalanced datasets.

Demerit: CatBoost can be computationally expensive and may require careful tuning of hyperparameters to avoid overfitting.

1. **Hist Extreme Gradient Boosting:** Hist Gradient Boosting is a variant of gradient boosting that uses histograms to speed up the training process and reduce memory usage while maintaining high accuracy. The max\_iter hyperparameter in Scikit-learn's implementation of HistGradientBoosting specifies the maximum number of boosting iterations (i.e., the maximum number of trees) to build during training. Increasing max\_iter can improve performance but may lead to overfitting and longer training times. The optimal value of max\_iter should be chosen to balance the trade-off between model performance and training time, which can be determined using techniques such as cross-validation. Increasing max\_iter improves performance to a certain point beyond which the model overfits.

The accuracy obtained at resource level 0 is **86.32%,** and at resource level 1 is **86.41%.**

Merit: HistGradientBoosting can be faster and more memory-efficient than traditional gradient boosting while maintaining high accuracy, making it suitable for large datasets.

Demerit: HistGradientBoosting may perform poorly with datasets with low instances or features.

1. **CTGAN:** Known as Conditional Tabular Generative Adversarial Network) is a type of generative adversarial network (GAN) explicitly designed for tabular data. It is a deep learning model that learns the underlying distribution of a dataset and generates synthetic data that is statistically similar to the original data.

It is useful for data augmentation, where additional synthetic data can be generated to increase the size of a dataset, as well as for preserving privacy by generating synthetic data that does not reveal sensitive information.

For our dataset, we increase the data by 2 lahks using CTGAN. The no. of data for each class after applying CTGAN is:

* 1. Benign – 127019
  2. Spyware – 44482
  3. Ransomeware – 43990
  4. Trojan – 43105

The highest accuracy obtained after applying CTGAN on the actual dataset for Resource level 0 is **91.43%** for hist extreme gradient boosting, while for Resource level 1, it is **92.17%** for hist extreme gradient boosting.

1. **SMOTE:** The Synthetic Minority Over-Sampling Technique creates synthetic examples of the minority class by interpolating between existing samples in the minority class. It selects one instance from the minority class and finds its k nearest neighbors in feature space. It then creates new synthetic examples by interpolating between the selected criterion and its neighbors.

The attribute amounts are – 1. Benign – 29298

* 1. Spyware – 10020
  2. Ransomeware – 9791
  3. Trojan – 9487

The highest accuracy obtained after applying SMOTE on the actual dataset for Resource level 0 is **89.26%** for Keras Neural Network, while for Resource level 1, it is **92.07%** for Random Forest.

Finally,

To obtain better accuracy, SMOTE was applied to the dataset generated by

CTGAN. It converted all the values in the “category” column to a total of 127019 records each, as 127019 is the highest no. of data that was obtained for the value Benign.

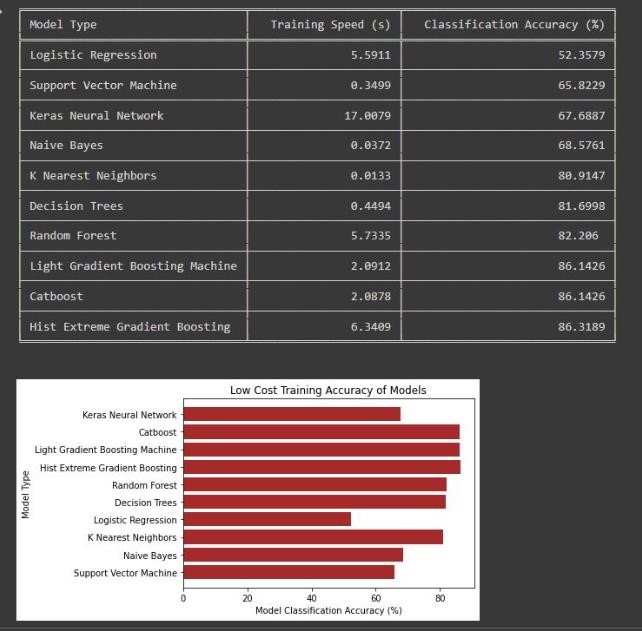
The Accuracy obtained after applying it is **94.27%** for Keras Neural Network at Resource Level 0, while the accuracy obtained at Resource Level 1 is **94.24%** for Random Forest.

# Results & Discussion

Our project aims to detect and identify the type of malware encountered. Thus, per our goal, we have used ten supervised and unsupervised models to classify and identify the various attribute values available under the feature labeled as “category.”

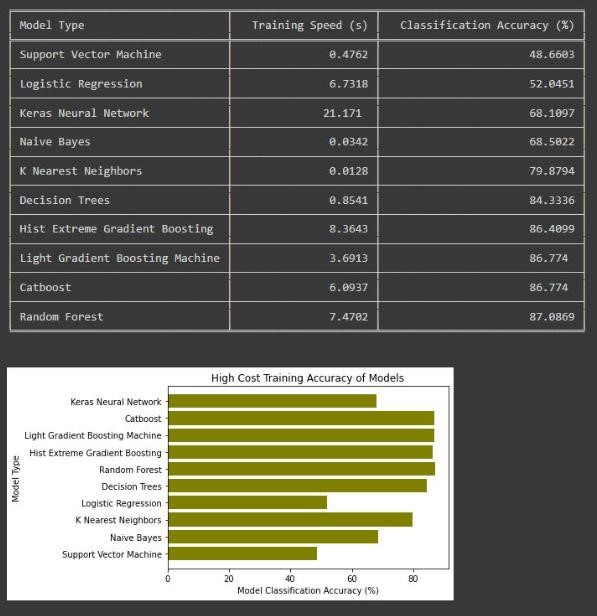
The models used are given below, with their training speed and accuracy values also mentioned.

*The figure below uses low-cost training, i.e., at resource level = 0.*



*Fig 1*

*The figure below uses high-cost training, i.e., at resource level = 1.*



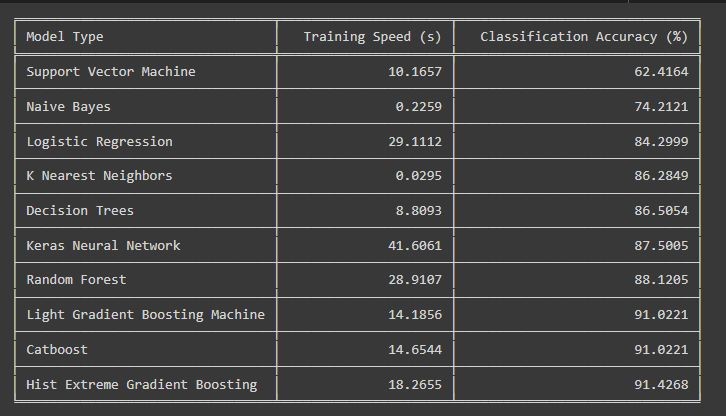
*Fig 2*

The predicted outputs obtained from each model were compared to the existing labels in the original dataset to get the accuracy percentage mentioned in the table.

The maximum accuracy obtained for prediction using ***resource level = 0*** is ***86.32%*** from the ***Histogram-Based Gradient Boosting*** model. While at ***resource level = 1***, it is ***87.09%*** from the ***Random Forest*** model. We have further tried to improve upon our model by using CTGAN and SMOTE on our dataset to increase the amount of data and treat data imbalance, respectively.

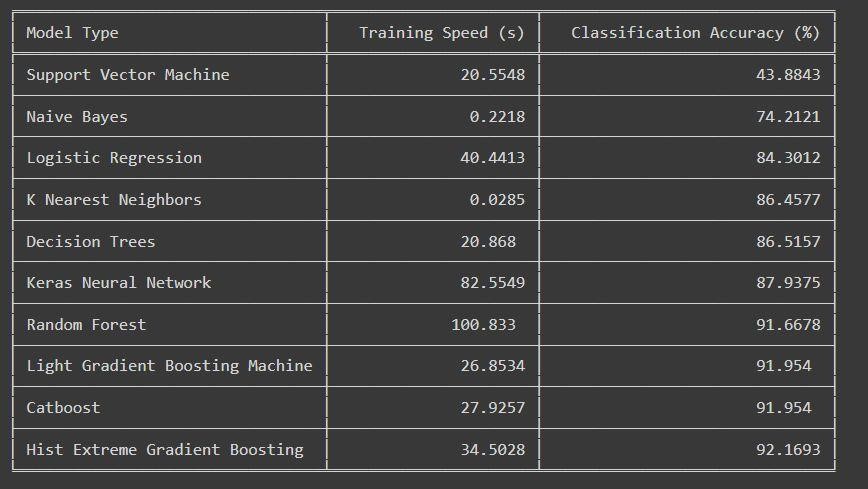
The obtained results are as follows –

*The figure below uses low-cost training, i.e., at resource level = 0 using CTGAN.*



*Fig 3*

*The figure below uses high-cost training, i.e., at resource level = 1 using CTGAN.*



*Fig 4*

*The figure below uses low-cost training, i.e., at resource level = 0 using SMOTE.*

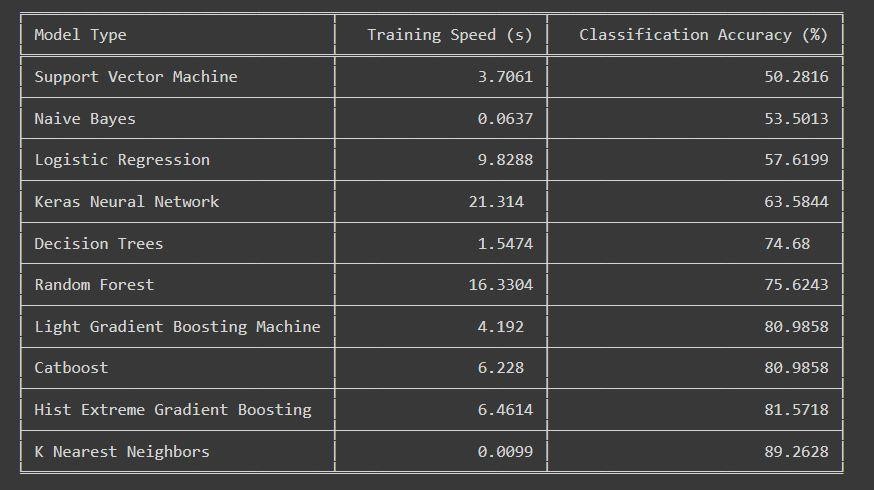
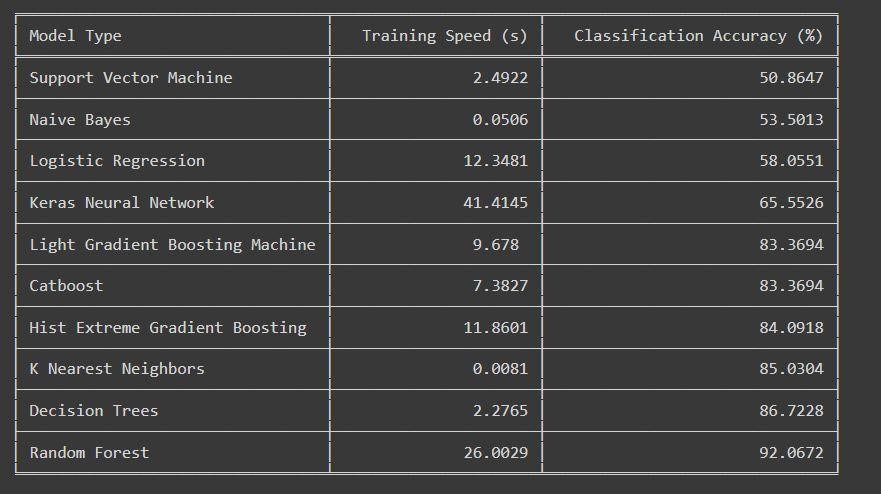


Fig 5

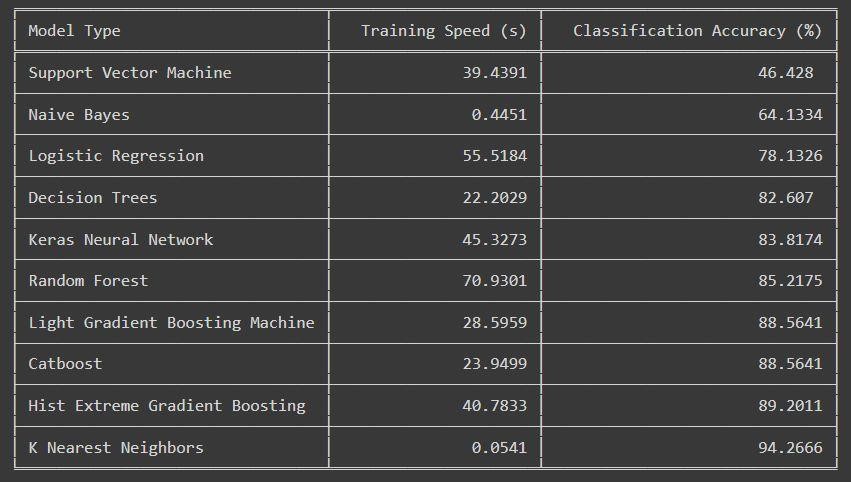
*The figure below uses high-cost training, i.e., at resource level = 1 using SMOTE.*



*Fig 6*

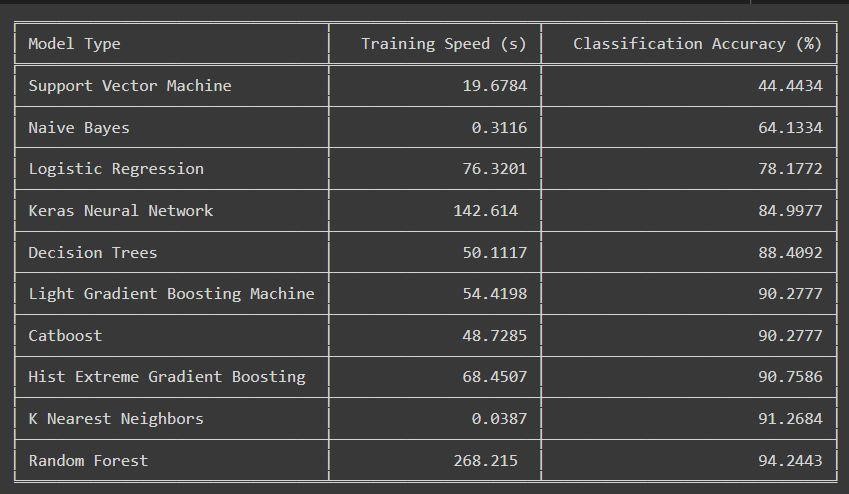
#### The figure below uses low-cost training, i.e., at resource level = 0 using both

*CTGAN & SMOTE.*



*Fig 7*

*The figure below uses high-cost training, i.e., at resource level = 1 using SMOTE.*



*Fig 8*

# References

1. [Garcia, F.C.C. and Muga II, F.P., 2016. Random forest for malware classification. arXiv preprint arXiv:1609.07770.](https://arxiv.org/abs/1609.07770)
2. [Ni, S., Qian, Q., and Zhang, R., 2018. Malware identification using visualization images and deep learning. Computers & Security, 77, pp.871-885.](https://www.sciencedirect.com/science/article/pii/S0167404818303481)
3. [Rhode, M., Burnap, P. and Jones, K., 2018. Early-stage malware prediction using recurrent neural networks. computers & security, 77, pp.578-594.](https://www.sciencedirect.com/science/article/pii/S0167404818305546)
4. [Alshamrani, S.S., 2022. Design and Analysis of Machine Learning](https://downloads.hindawi.com/journals/scn/2022/7611741.pdf)

[Based Technique for Malware Identification and Classification of Portable Document Format Files. Security & Communication Networks, 2022.](https://downloads.hindawi.com/journals/scn/2022/7611741.pdf)

1. [Halimaa, A. and Sundarakantham, K., 2019, April. Machine learningbased intrusion detection system. In 2019 3rd International conference on trends in electronics and informatics (ICOEI) (pp. 916-920). IEEE.](https://ieeexplore.ieee.org/abstract/document/8862784/)
2. [Choudhary, S. and Sharma, A., 2020, February. Malware detection & classification using machine learning. In 2020 International Conference on Emerging Trends in Communication, Control, and Computing (ICONC3) (pp. 1-4). IEEE.](https://ieeexplore.ieee.org/abstract/document/9117547/)
3. [Gavriluţ, D., Cimpoeşu, M., Anton, D. and Ciortuz, L., 2009, October. Malware detection using machine learning. In 2009 International multiconference on computer science and information technology (pp. 735-741). IEEE.](https://ieeexplore.ieee.org/abstract/document/5352759/)
4. [Kumar, S., Viinikainen, A. and Hamalainen, T., 2016, December. Machine learning classification model for the network-based intrusion detection system. In 2016 11th international conference for internet technology and secured transactions (ICITST) (pp. 242-249). IEEE.](https://ieeexplore.ieee.org/abstract/document/7856705/)
5. [Sethi, K., Chaudhary, S.K., Tripathy, B.K. and Bera, P., 2018, January. A novel malware analysis framework for malware detection and classification using a machine learning approach. In Proceedings of the 19th international conference on distributed computing and networking (pp. 1-4).](https://dl.acm.org/doi/abs/10.1145/3154273.3154326)
6. [Akhtar, M.S. and Feng, T., 2022. Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time. Symmetry, 14(11), p.2308.](https://www.mdpi.com/2073-8994/14/11/2308)
7. [Saxe, J. and Berlin, K., 2015, October. Deep neural network-based malware detection using two-dimensional binary program features. In 2015 10th international conference on Malicious and unwanted software (MALWARE) (pp. 11-20). IEEE.](https://ieeexplore.ieee.org/abstract/document/7413680/)
8. [Shijo, P.V. and Salim, A.J.P.C.S., 2015. Integrated static and dynamic analysis for malware detection. Procedia Computer Science, 46, pp.804811.](https://www.sciencedirect.com/science/article/pii/S1877050915002136)
9. [Mimura, M. and Ito, R., 2022. Applying NLP techniques to malware detection in a practical environment. International Journal of Information Security, 21(2), pp.279-291.](https://link.springer.com/article/10.1007/s10207-021-00553-8)
10. [Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D. and Liu, H., 2020. A review of Android malware detection approaches based on machine learning. IEEE Access, 8, pp.124579-124607.](https://ieeexplore.ieee.org/abstract/document/9130686/)
11. [Mezina, A. and Burget, R., 2022, October. Obfuscated malware detection using the dilated convolutional network. In 2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT) (pp. 110-115). IEEE.](https://ieeexplore.ieee.org/abstract/document/9943443/)
12. [Li, Y., Liu, Z., Guan, X., Wang, Z., Guo, X. and Wang, S., 2022, June.](https://ieeexplore.ieee.org/abstract/document/10048243/)

[Hierarchical Obfuscation Malware Detection Method Based on Deep Learning. In EEI 2022; 4th International Conference on Electronic Engineering and Informatics (pp. 1-4). VDE.](https://ieeexplore.ieee.org/abstract/document/10048243/)

1. [Zhu, J., Jang-Jaccard, J., Singh, A., Watters, P.A. and Camtepe, S., 2021. Task-aware meta-learning-based siamese neural network for classifying obfuscated malware. arXiv preprint arXiv:2110.13409.](https://arxiv.org/abs/2110.13409)
2. [Roseline, S.A., Sasisri, A.D., Geetha, S. and Balasubramanian, C., 2019, October. Towards efficient malware detection and classification using multilayered random forest ensemble technique. In 2019](https://ieeexplore.ieee.org/abstract/document/8888406/)

[International Carnahan Conference on Security Technology (ICCST) (pp. 1-6). IEEE.](https://ieeexplore.ieee.org/abstract/document/8888406/)

1. [Baig, D., Khan, M.U., Dancey, D., Abbas, A., Ali, M. and Nawaz, R.,](https://ieeexplore.ieee.org/abstract/document/9701433/)

[2021, December. Malware Detection and Classification along with](https://ieeexplore.ieee.org/abstract/document/9701433/)

[Trade-off Analysis for Number of Features, Feature Types, and Speed. In 2021 International Conference on Frontiers of Information Technology (FIT) (pp. 246-251). IEEE.](https://ieeexplore.ieee.org/abstract/document/9701433/)

1. [Pachhala, N., Jothilakshmi, S. and Battula, B.P., 2021, October. A comprehensive survey on identification of malware types and malware classification using Machine Learning Techniques. In 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC) (pp. 1207-1214). IEEE.](https://ieeexplore.ieee.org/abstract/document/9591763/)