Machine Learning Based Malware Classification based on API call Histogram

Abhisht Joshi [a], Harsh Patel [b], Jainam Shah [c], Snehal Bhole [d]

[a] Abhisht Joshi, University Of Ottawa, Ottawa, Canada, ajosh053@uottawa.ca

[b] Harsh Patel, University Of Ottawa, Ottawa, Canada, hpate158@uottawa.ca

[c] Jainam Shah, University Of Ottawa, Ottawa, Canada, jshah082@uottawa.ca

[d] Snehal Bhole, University Of Ottawa, Ottawa, Canada, sbhol010@uottawa.ca

Abstract

With the increase in the variety of malware samples, malware classification has become important. Static, dynamic, and hybrid features are extracted to classify the different malware samples into different malware families. In this work, we focus on classifying malware samples into 9 classes. The task aims to classify API call histogram input values into different malware classes. API call frequencies are the static representation of the behaviour of malware classes. The dataset was gathered from the International CyberSecurity Data Mining Competition (CDMC) 2022, provided by Paul Black. The primary approach to the problem consists of exploratory data analysis, training different machine learning models (Support Vector Machine, Multi-layer Perceptron, CatBoost, Random Forest) and comparing their training with a baseline model (Decision Tree). To evaluate the performance of the models on the test data (unlabelled), we use Cohen's Kappa score. The score for our models shows the reliability of the models.

Keywords: *Malware classification, API call histogram, Cohen's Kappa Score, Cyber Security,*

1. Introduction

In this report, we will explore the use of machine learning techniques to classify malware based on the API call histogram. We will first provide an overview of machine learning and its application in malware detection. Then, we will discuss the various methods used to create and analyze the API call histogram. Finally, we will present the results of our classification experiments and discuss the implications for future research in this field.

1.1. Malware

Malware is a term used to describe software that is designed to perform malicious actions on a computer or network without the user's knowledge or consent[1]. Malware can take many different forms, including viruses, worms, Trojans, ransomware, and spyware. Malware is often spread through email attachments, infected websites, or other methods that allow it to be delivered to and executed on a victim's computer

1.2. Malware API call sequence

Malware API call sequence refers to the specific sequence of API (Application Programming Interface) calls that are made by malware when it is running on a computer or network. API calls are a normal and necessary part of the operation of most software, including both legitimate and malicious software. They allow the software to access and manipulate system resources, such as files, processes, and network connections.

Malware may use any API calls that are available to it, depending on its intended functionality and the environment in which it is running. For example, a piece of malware that is designed to steal sensitive information from a computer might make API calls to access and read the contents of certain files or directories on the system, or to exfiltrate this information to a remote server.

Analyzing the sequence of API calls made by malware can potentially be useful for understanding its behaviour and identifying patterns or trends in its operation. However, it is important to note that the presence of API calls alone is not necessarily indicative of malware, and other factors, such as the overall behaviour of the software and the context in which it is running, must also be considered.

1.3. Dataset

The CDMC (Computational Intelligence in Data Mining and Cybersecurity) 2022 competition includes a dataset of malware samples that were built from samples provided by Abuse.ch [2]. The features for each sample were extracted using dynamic analysis in a Cuckoo sandbox and consisted of a histogram showing the count of different API (Application Programming Interface) calls made by the sample. The dataset contains a total of 9, malware classes. The data set consists of 2 files training and testing file training files have 537 labelled sample and test files contains 134 unlabelled data. The API sequence is presented through the 208 API call sequence. Labels are contained in the first column of the training data. The dataset’s structural information is shown in Table 1.

Dataset	File	Samples(Rows)	Dimension(columns)
Training	train_features.txt	537	209(1st column is the class label)
Testing	Test_feature.txt	134	208

Table 1: Dataset Structure

2. Literature Review

The categorization of malware has been the subject of several research, many of which have made use of machine learning algorithms. In [3], the authors used a number of machine learning methods, such as decision trees, KNN and support vector machines, to sort malware samples based on their static and dynamic properties. They found that the decision tree algorithm worked the best, with an accuracy of 97.3%. The paper [4] talks about ML algorithms like SVM and how to set up an artificial neural network. It also discusses how each method is implemented and works on various datasets. Chen Li and Junjun Zheng [5] have implemented Recurrent Neural Networks especially LSTM and GRU, for malware classification. In the first paper [6], the authors used the LSTM model, but the model was not simply sent a series of plain API calls. They developed a novel strategy known as the "sliding window technique." They split the API sequence first before converting it to a digital sequence. The complete API sequence is translated to a number, and each API function has its own individual number assigned to it. The N-gram approach will generate a lot of segments, but the overall location map of the many N-gram segments won't accurately depict each segment's true meaning. The truly useful segment information will be buried, however. They then applied the same category tag to the API fragments produced by the sample. A specific digital slice serves as the representation of the API fragment. These slice libraries with various labels are produced by separating the slices. In contrast to the information entropy method used in [7], the author uses a deduplication method to reduce the number of identical slices in the libraries by a factor of two to two.A novel dataset that has never existed in this sector before, API calls from the Windows Operating system were revealed by Catak et al. [8]. To categorise malware families, they suggested a single-layer LSTM and a two-layer LSTM in their work.

According to the paper [9], earlier techniques do not produce reliable findings due to the rapid development of novel malware. Data is labelled for Supervised Learning, and massive amounts of labelled data are required to detect malware accurately. It is impractical to annotate vast amounts of data with the limited expert resources available to label data. Furthermore, the supervised model's generalization potential for malware detection is

limited. The authors employ semi-supervised learning to use unlabeled data in this strategy. The authors suggested a new colSVM method with collaborative training and a semi-supervised approach. The approach uses Independent Component Analysis (ICA) to pre-process the dataset and divides the feature into two unconnected components. Later, the SVM classifier is applied to the two portions. A modest amount of data can be used to detect malware using this methodology. A feedforward neural network was suggested by Raff et al. [10] to categorize dangerous executables from raw byte sequences. The strategy addressed some drawbacks of the more popular byte n-gram approach, including brittle features and an excessive emphasis on the PE-Header as significant data.

Some work that involves the implementation of Natural Language Processing (NLP) was researched,\ to distinguish between malware and benign software, Nagano et al. [11] employed a combination of static analysis, machine learning classifiers, and natural language processing. Static analysis of malware and benign programme execution files is used to gather data about DLL import, assembly code, and hex dump, which is then processed by the PV-DBOW model to provide feature vectors. Then, to assess and categorise malware, these vectors are fed into SVM and k-nearest neighbour classifiers. The classification accuracy solely on DLL information was the lowest throughout all experiments (approximately 85.1% and 92.1% for SVM and k-NN, respectively),

3. Research Objectives

The objective of this report is to evaluate the effectiveness of using machine learning algorithms to classify malware API calls based on their potential maliciousness. This will be achieved through the analysis and comparison of various machine learning models, including decision trees, random forests, support vector machines, MLP and Cat-Boost to determine which model is most accurate in detecting and categorizing malware API calls. The report will also explore the impact of different feature selection techniques on the classification performance, and provide recommendations for the optimal model and feature selection approach for classifying malware API calls using machine learning.

4. Methodology

4.1 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an important step in the machine learning process that involves analyzing and understanding the data that will be used to train a model. It is typically performed before the actual

modelling step, and it helps to identify patterns, trends, and relationships in the data, as well as to identify any potential issues or problems that may need to be addressed before the model can be trained[12]. The goal of EDA is to get a better understanding of the data and to identify any potential problems or issues that could affect the model's performance.

4.1.1 Class Distribution

Class distribution refers to the distribution of different classes or categories within a dataset. It is important to consider class distribution when building a model, as imbalanced class distributions can impact the model's performance. For example, if one class is much more prevalent than the other classes, the model may be more likely to predict the more prevalent class, leading to poor performance in the less prevalent classes.

The given figure[1] shows data separated into nine different classes, each representing a different malware family. It is clear from the figure that the classes do not form distinct clusters, meaning that it may be difficult to accurately classify them. To address this issue, one common approach is to use sampling techniques to balance the class distribution in the training data.

There are several sampling techniques that can be used, including oversampling (adding more samples of the underrepresented classes) and undersampling (removing samples from the overrepresented classes). It is also possible to use a combination of oversampling and undersampling, known as hybrid sampling. In addition to balancing the class distribution, it is also important to ensure that the sampling method used is representative of the overall population. For example, if the dataset is collected from a specific group of people, it is important to ensure that the sampling method used to balance the class distribution does not introduce bias or distort the overall distribution of the data.

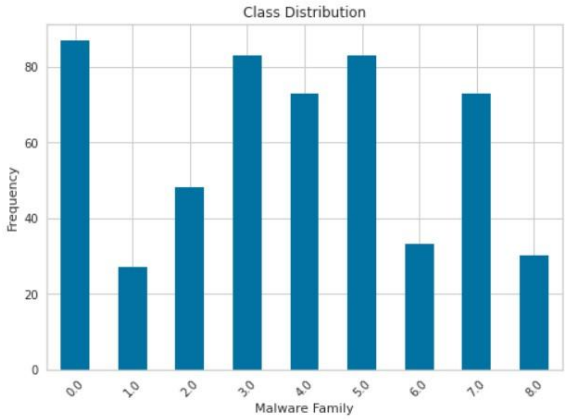

Figure 1: Frequency Distribution Plot

4.1.2 T-SNE Plot

T-SNE, or t-distributed stochastic neighbour embedding, is a machine-learning technique used for visualizing high-dimensional data in a lower-dimensional space. It is particularly useful for visualizing clusters or patterns in a dataset that may not. Once it has been installed, malware can perform a wide range of malicious actions, such as stealing sensitive information, corrupting or deleting data, or using the infected system as part of a botnet for spamming or other nefarious purposes.

T-SNE works by minimizing the divergence between the probability distributions of the high-dimensional data and the low-dimensional data, using a cost function called the Kullback-Leibler divergence. It is a non-linear technique, which means that it can capture complex relationships between data points that may not be captured by linear techniques such as PCA (principal component analysis)[16].

One disadvantage of T-SNE is that it can be computationally expensive, especially for large datasets. It is also sensitive to the choice of parameters, such as the perplexity and learning rate, which can affect the final visualizations. Despite these limitations, T-SNE has become a popular tool for exploring and visualizing high-dimensional datasets in a variety of fields including machine learning, data science, and biology[13].

Here, we have used T-SNE to reduce the dimensionality of the data and plotted the resulting visualizations in both two dimensions and three dimensions. figure[2], shows the T-SNE plot for the given dataset for the 9 malware families considered.

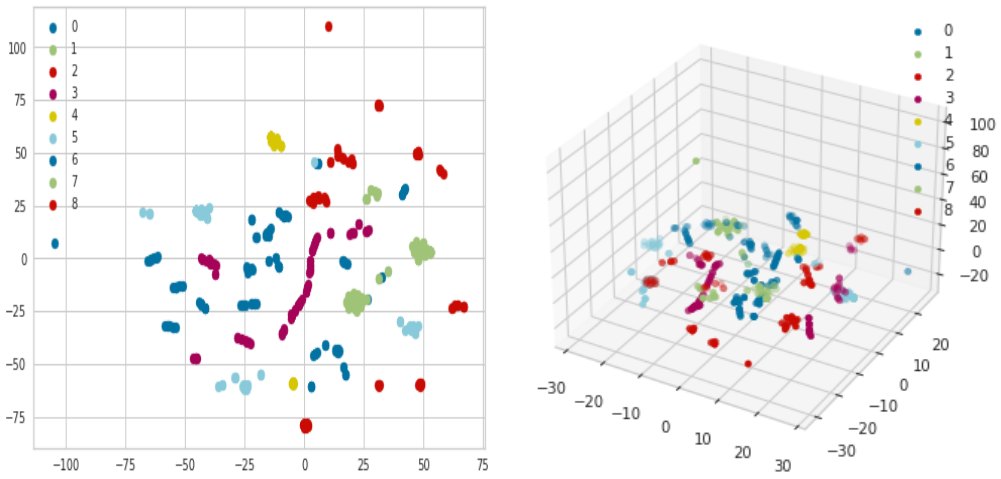

Figure 2: T-SNE plot for distinct Malware

4.1.3 Range of Values

When working with data, it is important to consider the range of values and magnitude of the data. The range of values refers to the difference between the minimum and maximum values in the data. The magnitude refers to the size or scale of the values. Scaling is a process used to adjust the range and magnitude of the data so that it can be more easily compared and processed by machine learning algorithms.

There are several different methods for scaling data as listed below:

- Min-Max Scaling: This method scales the data to a specific range, such as 0 to 1 or -1 to 1. It is useful when the data has a skewed distribution and the outliers need to be preserved.
- Standardization: This method scales the data so that it has a mean of 0 and a standard deviation of 1. It is useful when the data is approximately normally distributed and the outliers do not need to be preserved.
- Normalization: This method scales the data so that it has a minimum value of 0 and a maximum value of 1. It is useful when the data has a skewed distribution and the outliers do not need to be preserved.

In our dataset, we observed that the API call frequency values have a wide range of frequencies. To ensure that the API call frequency values can be effectively used as input for a machine-learning model, we applied min-max scaling to the data. This scaling method adjusts the range and magnitude of the data so that it can be more easily compared and processed by the model.

4.1.4 Number of Features

It is common to use a large number of features when training a machine learning model, as more features can potentially provide the model with more information about the data. However, it is important to consider the quality and relevance of the features being used. Using too many irrelevant or redundant features can increase the complexity of the model and lead to overfitting, which is when the model performs poorly on unseen data.

To address this issue, it is often helpful to perform feature selection, which is the process of selecting a subset of the most relevant and informative features for use in the model.

There are several different methods for feature selection, as listed below:

- Filter methods: These methods use statistical measures to select the most relevant features.

- Wrapper methods: These methods use the performance of the model to select the most relevant features.
- Embedded methods: These methods select the most relevant features as part of the model training process.

It is important to carefully evaluate the performance of the model using different subsets of features to determine the optimal number and combination of features to use. "For the purpose of training our machine learning model, we have identified a total of 208 different API calls in our dataset to use as features. These API calls represent various characteristics of the data and provide the model with information that can be used to make predictions.

4.2 Feature Engineering and Feature Pre-Processing

Feature engineering and feature preprocessing are techniques used to prepare and transform the data before it is fed into a machine-learning model. Feature engineering involves creating new features or modifying existing ones to make them more useful for a specific task. This can involve tasks such as aggregating existing features, creating new features based on domain knowledge, or selecting a subset of the most relevant features.

Feature preprocessing involves cleaning, scaling, and normalizing the data to ensure that it is in a suitable format for the model. This can include tasks such as handling missing values, encoding categorical variables, and scaling the data to have zero mean and unit variance. Both feature engineering and feature preprocessing are important steps in the machine learning process, as they can have a significant impact on the performance of the model.

4.2.1 Oversampling Technique: SMOTE

We have performed SMOTE to balance the class.SMOTE (Synthetic Minority Over-sampling Technique) is a popular algorithm used to address imbalanced datasets in machine learning classification problems. It works by generating synthetic samples of the minority class in order to balance the class distribution.

Class imbalance occurs when the number of samples in one class is significantly smaller than the number of samples in the other class. This can be a problem because some machine learning algorithms are sensitive to imbalanced class distributions and may not perform well on imbalanced datasets.

SMOTE works by selecting a minority class sample and finding its k-nearest neighbours. It then generates synthetic samples by selecting a random point along the line connecting the sample and each of its neighbours and adding the synthetic samples to the minority class. This process is repeated until the minority class has the same number of samples as the majority class.

SMOTE has been shown to be effective at improving the performance of machine learning classifiers on imbalanced datasets. However, it is important to evaluate its performance on your specific dataset and consider other approaches as well.

4.2.2 Scaling: Min-Max Scaler

To normalize the data, we applied Min-Max Scaler. The Min-Max Scaler is a preprocessing technique that scales a feature's values to lie between a given minimum and maximum value, usually 0 and 1. It transforms the data by subtracting the minimum value and dividing the result by the range of the data. The transformed data will have values between 0 and 1, with 0 corresponding to the minimum value and 1 corresponding to the maximum value.

The Min-Max Scaler is often used when the data distribution is not known, or when the data follow a skewed distribution. It is also used when the algorithm being used is sensitive to the scale of the input data, such as algorithms that use distance measures, like K-Nearest Neighbors. As we have performed SVM which uses euclidian distance.

4.2.3 PCA (Principal Component Analysis)

To reduce the dimensionality and reduce the number of features we have performed PCA. Principal Component Analysis (PCA) is a popular dimensionality reduction technique in machine learning that is used to reduce the number of variables in a dataset while still retaining as much of the variance as possible[16]. It does this by projecting the data onto a new set of axes, known as principal components, that are ranked in order of importance. PCA is often used as a preprocessing step for machine learning algorithms, as it can help reduce the computational cost and improve the performance of the model. It can also be used for data visualization, as it can help to reveal patterns and relationships in the data that may not be immediately apparent. There are several variations of PCA, such as Kernel PCA and Incremental PCA, which can be used in different situations depending on the characteristics of the data and the needs of the application.

4.2.4 Feature Selection: Mutual Information Gain

In feature selection to select the best feature we have performed a Mutual Information Gain. Mutual Information Gain (MIG) is a measure of the mutual dependence between two variables. It quantifies the amount of information that one variable provides about the other. MIG is often used in feature selection, where it is used to identify the most relevant features for a given task.

MIG can be calculated using the formula:

$$MIG(X, Y) = \sum x \sum y \; p(x,y) \log[p(x,y)/p(x)p(y)]$$

where X and Y are the two variables, p(x) is the probability of x occurring, p(y) is the probability of y occurring, and p(x,y) is the joint probability of x and y occurring.MIG ranges from 0 (no mutual dependence) to 1 (complete mutual dependence). A high MIG value indicates that the two variables are highly dependent on each other, while a low MIG value indicates that the variables are independent. MIG is a useful measure because it takes into account the probability distribution of the variables, rather than just the correlation between them. This makes it more robust to non-linear relationships and allows it to capture more complex dependencies in the data.

4.3 Parameter Tuning (Phase 1)

In phase one, we are performing a split on data using the hold-out method. The hold-out approach includes slicing the data into many pieces, utilising one piece for training, one piece for testing, and the remaining piece for validation. For phase one we are splitting data in 80:20 ratio. The decision tree is used as the baseline model. The decision tree is a rule-based classification, prominently used in Machine Learning Classification projects. Although, Decision trees are very effective tools, but may not be an appropriate alternative in all circumstances. It is simple to handle a combination of numerical and non-numerical data and to analyse data using a Decision tree[1]. Additionally, it only needs a little preparation like, cleaning and preprocessing before usage. On the contrary, they are not suitable for continuous variables. Predictive Analysis can become a cumbersome task when working with a decision tree.

The flowchart given in figure[3] depicts the approach used in phase 1. Data is scaled for the development of an effective model after being divided into train and test sets. Since each malware is equally significant, SMOTE is

being used to oversample. In order to identify a smaller representation of the data and minimise the number of input characteristics, the principle component analysis is applied to the scaled data from selected input features.

In PCA, every feature has a strong correlation with every other feature.PCA analysis was carried out with 0.99 covariances. Based on this, the entire dataset was represented into 34 input features. After performing Principal Component Analysis (PCA), we used Mutual Information Gain (MIG) to select the best features. However, there was not much difference in the results, so we decided to consider all of the features in our analysis. Finding the ideal parameters to create the best-performing models was then the next stage.GridSearchCV was used. To determine the model's ideal parameters, hyperparameter tuning is done using GridSearchCV. The values of the parameters used have a considerable impact on how a model behaves. The model selection package of Scikit-Learn contains the GridSerachCV function. All possible value combinations are tested using GridSearchCV, which then uses the cross-validation approach to assess the models for each combination. Then, after receiving accuracy for each combination, we select the one that performs the best.

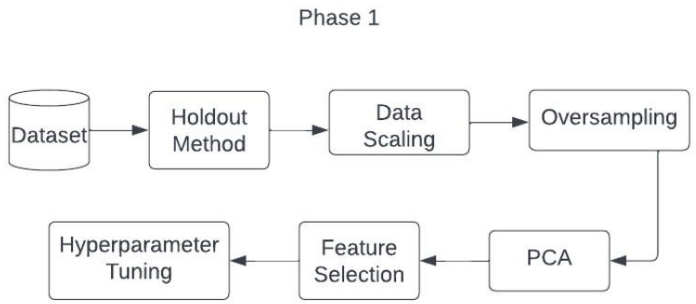

Figure 3: Phase 1 Workflow

4.4 Model Selection (Phase 2)

In phase 2, we are using stratified k-fold Cross Validation. stratified k-fold cross-validation is generally preferred in cases where the class distribution is imbalanced, as it helps to reduce the impact of any biases due to uneven class distributions.

Stratified 5-fold cross-validation is a method of evaluating the performance of a machine learning model by dividing the data into 5 folds, with each fold representing approximately the same proportion of each class as the overall dataset. On each iteration of the 5-fold cross-validation procedure, a different fold is used as the test set while the remaining folds are used as the training set.

In this step, we are using stratified 5-fold cross-validation as part of a pipeline that includes min-max scaling, SMOTE, and PCA. Min-max scaling is a method of scaling the data so that all the features are in the same range, usually between 0 and 1. SMOTE is an algorithm that generates synthetic samples of the minority class in order to balance the class distribution in a dataset. PCA is a dimensionality reduction technique that projects the data onto a new set of axes that are ranked in order of importance.

By performing stratified 5-fold cross-validation in combination with these other preprocessing steps, we were able to evaluate the performance of our model while also taking into account the impact of scaling and balancing the data, as well as reducing the number of features. This helps to ensure that the model is robust and generalizes well to new data. The workflow of the phase is shown in Figure 4, where the scaling, oversampling, and PCA are included through a pipeline, which helps to apply each function for each iteration differently.

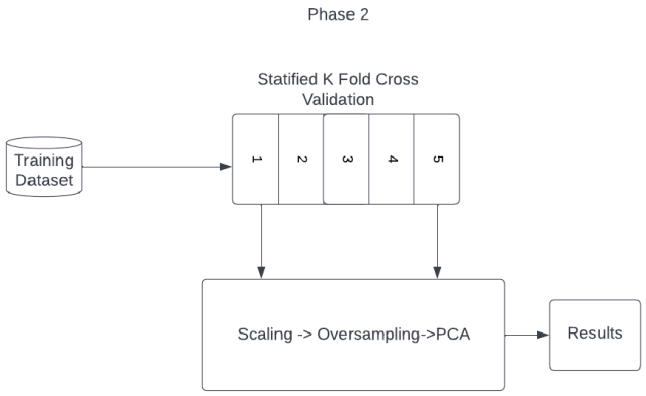

Figure 4: Phase 2 Workflow

4.4 Model Evaluation (Phase 3)

The CDMC 2022 Hackathon has not provided labels for the test data. The evaluation metrics used in supervised learning are non-tangible. So to evaluate the models on unseen data, we have to use Cohen's Kappa score to measure the reliability of the models on the classification task (qualitative items). This phase mainly includes two parts to justify the reliability of the models.

a. Two best classifiers (Random Forest and Support Vector Classifier) - This evaluation shows the confidence of the two models and to which degree they agree with each other. A high score in this shows that are models are equally confident.

b. Best classifier and bad model (Support Vector Classifier /Random Forest Classifier and Majority Class Classifier) - This evaluation checks the agreement of the best classifier with a bad classifier to see justify that the selected model is not a bad classifier. So a low cohen kappa score shows the disagreement between the selected model with a bad classifier.

5. Results

The models (Decision Tree, Random Forest, Multi-Layer Perceptron, Support Vector Classifier, and Cat Boost) employed are used to classify malware classes on the basis of API call histogram. These are evaluated against different metrics (accuracy, f1 score, precision, recall, and cohen kappa score).

5.1 Model Selection

For the selection of the model we take into consideration the performance of the models during the hold-out method and cross-validation. For the holdout method, we do have confusion matrices for the models.

Figure[5] given shows the confusion matrix for four different models: Decision Tree, Random Forest, Cat Boost, and MLP. A classification model's (or "classifier's") performance on a set of test data for which the true values are known is frequently described using a confusion matrix. By contrasting the predicted class labels with the actual class labels, it enables you to visualise the performance of a classifier. For each model, a separate confusion matrix is plotted to illustrate the performance of the model on the classification task.

In the hold-out stage, the results of the confusion matrix show that all of the models performed very well, with almost all of the classifiers achieving perfect classification. This means that the models were able to accurately predict the class labels for the majority of the instances in the dataset.

5.2 Cross-Validated graphs

The graphs in Figure[6,7,8,9] show the performance of different models on the accuracy, precision, recall and f1-score respectively in each iteration of 5-Fold cross-validation, we can observe for each fold test data we get results in the range of 0.95 - 1.

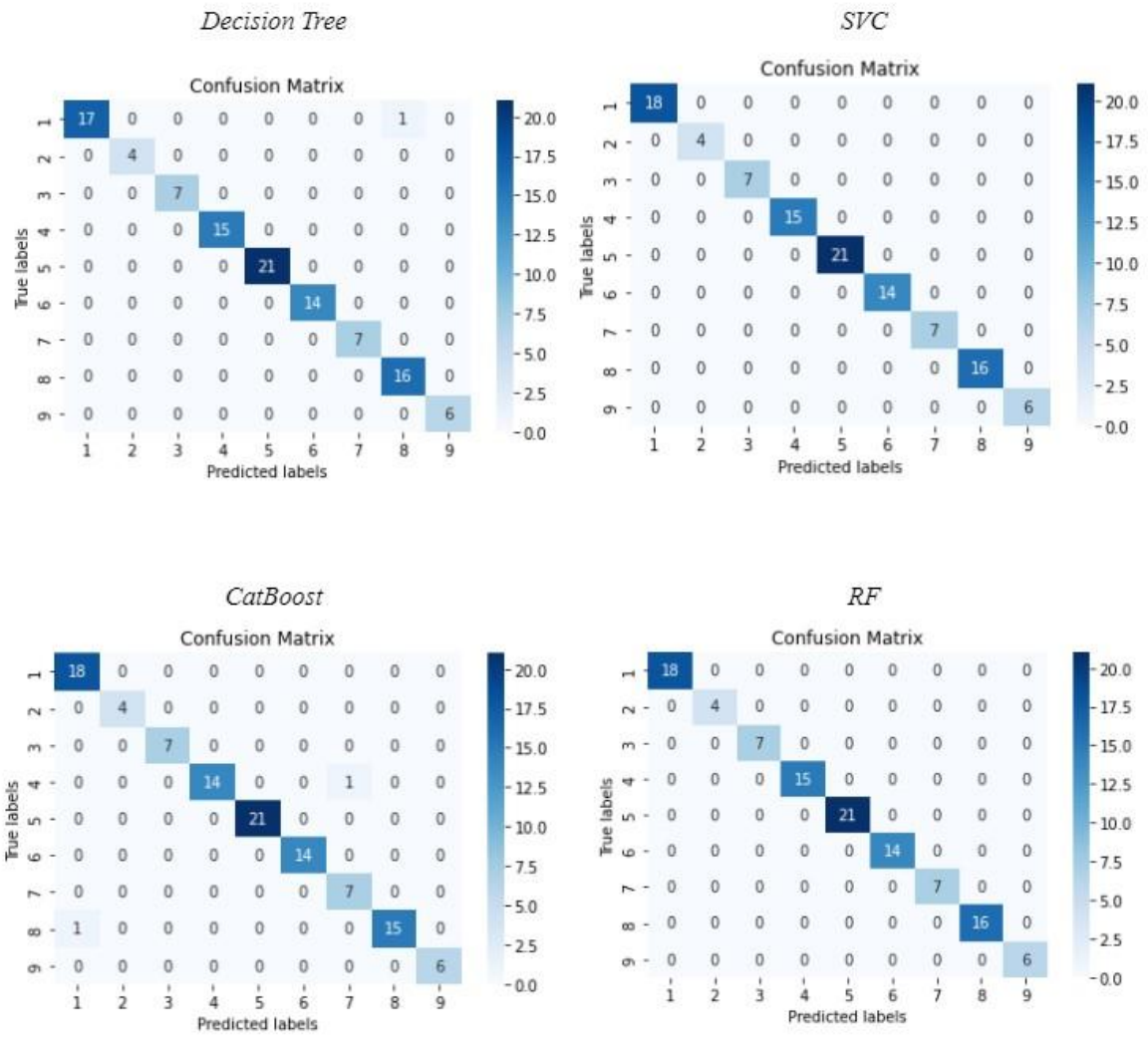

Figure 5: Confusion Matrix w.r.t distinct Algorithm

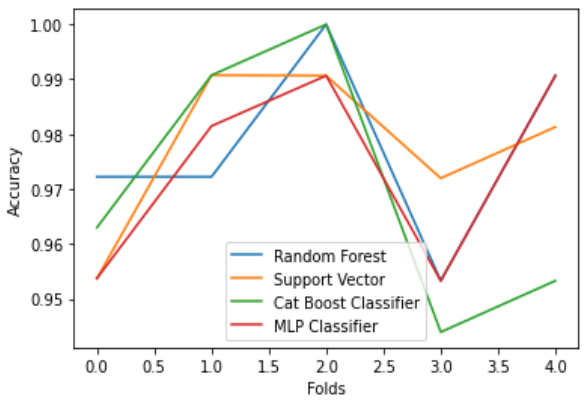

Figure 6: Accuracy for distinct Algorithms

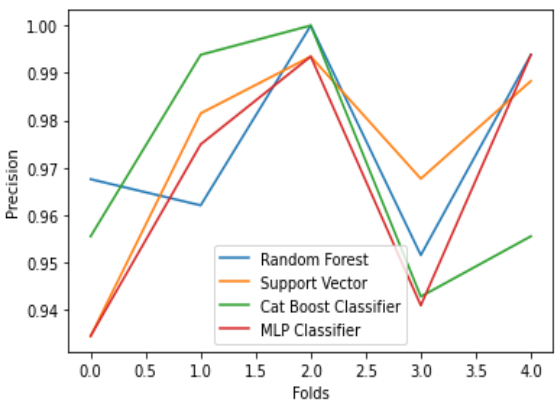

Figure 7: Precision for distinct Algorithms

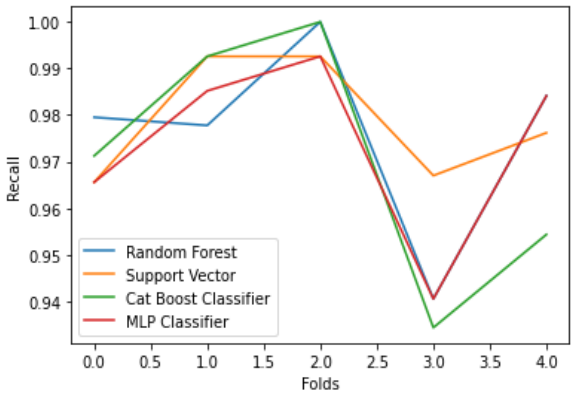

Figure 8: Recall for distinct Algorithms

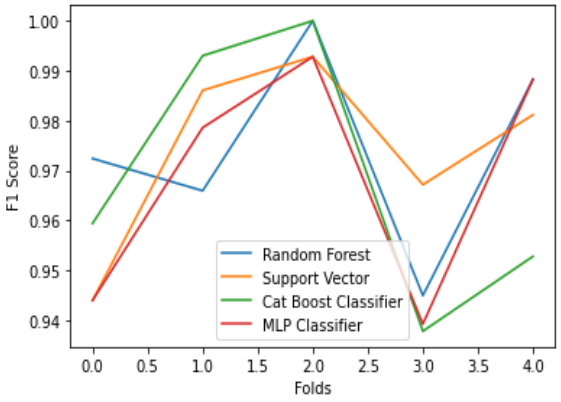

Figure 9: F1 Score for distinct Algorithms

All the four metrics(accuracy, f1 score, precision, and recall) for the five models are evaluated and the results are shown in Table 2.

Classifiers	**F1 Score**	**Precision**	**Recall**	**Accuracy**
Decision Tree (Average)	0.966	0.9658	0.9708	0.9701
Decision Tree (Holdout)	0.9934	0.9934	0.9938	0.9907
MLP (Average)	0.9666	0.9658	0.9722	0.9721
MLP (Holdout)	0.9931	0.9941	0.9925	0.9907
SVC (Average)	0.9742	0.9730	0.9787	0.9776
SVC (Holdout)	1	1	1	1
CatBoost (Average)	0.9677	0.9667	0.9708	0.9702
CatBoost (Holdout)	0.9821	0.9802	0.9856	0.9814
RandomForest (Average)	0.9733	0.9736	0.9763	0.9776
RandomForest (Holdout)	1	1	1	1

Table 2: Performance Measure for Distinct Algorithms

For each model we performed the holdout method (GridsearchCV) as well as an average of metrics of the results obtained by K-Fold Cross Validation. From the above results, we can clearly see that **SVC** (Support Vector Classifier) and **RFC** (Random Forest Classifier) have attained the highest F1 Score of 1 in the holdout method, similarly, when we compare the average accuracy, **SVC** and **RFC** attained the highest accuracy of

97.76% accuracy. Thus, we can conclude that **SVC** and **RFC** have outperformed other algorithms in terms of F1 score and accuracy metrics and thus we can say that they are the best classifier for this research.

5.3 Evaluation of Test Data

The following Table 3 shows the Cohen's Kappa score amongst the classifiers that attained the best results on the dataset as well as the Majority class classifier and the best models.

Models	**Majority Class Classifier and SVC**	**Majority Class Classifier and Random Forest**	**SVC and Random Forest**
Cohen Kappa Score	0	0	0.9742

Table 3: Cohen's Kappa Score

6. Conclusion

In this work, the goal is to classify malware samples into 9 different classes based on API call histogram input values. After completing all necessary exploratory data analysis (EDA) and feature engineering steps, we used the hold-out method to find the best parameters for our models. We then applied these parameters in a cross-validation step to evaluate the performance of various models. Our results showed that Random Forest and Support Vector Classifier (SVC) outperformed other models in terms of accuracy, with SVC producing the best results in terms of the F1-score. We also used Cohen's Kappa test to assess the reliability of our models, comparing the best-performing models (SVC and Random Forest) to each other and to the worst-performing classifier.

The Cohen's Kappa score between the Support Vector Classifier (SVC) and Random Forest Classifier (RFC) was achieved at 0.9742, which basically implies that both models are in agreement with each other and are confident, to determine whether the best model is correctly predicting labels or whether the best classifiers are producing high Cohen's kappa scores simply because neither model is correctly predicting. To reject that hypothesis we compared the Cohen's Kappa score between the Majority Class Classifier and the best classifier i.e., Majority Class Classifier and Support Vector Classifier and Majority Class Classifier and Random Forest Classifier which comes out to be **Zero**. Cohen's kappa score of **zero** demonstrates the disagreement between the chosen model and the bad classifier, proving that both models are reliable.

7. Future Work

There are many directions in which future work on malware classification could be focused. Finding traits that may be utilized to distinguish between various types of malware is one of the main problems in malware classification. The focus of future research could be on creating more efficient techniques for extracting information from malware samples. Also, implementing other ensemble learning algorithms apart from the algorithms used in the project. The development of ensemble learning methods for malware classification in future studies could contribute to increasing the resilience and dependability of the classifier. Also, future work could explore the use of transfer learning techniques and Adversarial learning for malware classification, improve the classifier's ability to detect novel or previously unseen malware and to improve the classifier's ability to resist attacks and evade detection.

8. REFERENCES

[1] https://www.paloaltonetworks.com/cyberpedia/what-is-malware

[2] https://www.csmining.org/cdmc2022/index.php?id=5.

[3] S. Singh, G. Kaur, and H. Kaur, "Malware classification using machine learning techniques," International Journal of Advanced Research in Computer Science, vol. 8, no. 1, pp. 30-35, 2017.

[4] C.L.P. Chen, Y.S. Chen, and Y.M. Huang, "A survey of machine learning techniques for malware detection," Computers & Security, vol. 66, pp. 18-41, 2017.

[5] Chen Li, Junjun Zheng, "API Call-Based Malware Classification Using Recurrent Neural Networks," in Journal of Cyber Security and Mobility, 2021: Vol 10 Iss 3

[6] Xin Ma, Shize Guo, Wei Bai, Jun Chen, Shiming Xia, Zhisong Pan, "An API Semantics-Aware Malware Detection Method Based on Deep Learning", Security and Communication Networks, vol. 2019, Article ID 1315047, 9 pages, 2019.

[7] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," Frontiers of Information Technology & Electronic Engineering, vol. 18, no. 9, pp. 1336–1347, 2017.

[8]F. O. Catak, A. F. Yazi, and O. Elezaj, "Deep learning based Sequential model for malware analysis using Windows exe API Calls," *PeerJ Computer Science*, vol. 6(81), July 27, 2020.

[9] Kai Zhang, Chao Li, Yong Wang, Xiaobin Zhu, Haiping Wang, Collaborative Support Vector Machine for Malware Detection, Procedia Computer Science, Volume 108,2017, Pages 1682-1691, ISSN 1877-0509

[10] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," *Proceedings of the Workshops 32nd AAAI Conf. Artif. Intell.*, pp. 268–276, 2018.

[11] Y. Nagano and R. Uda, "Static analysis with paragraph vector for malware detection", *IMCOM '17 Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, 2017.

[12] https://www.ibm.com/cloud/learn/exploratory-data-analysis

[13]https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1#:~:text=What%20is%20t%2DSNE%3F,in%20a%20high%2Ddimensional%20space.

[14] https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/

[15] https://www.csmining.org/cdmc2022/index.php?id=5.

[16] https://towardsdatascience.com/understanding-pca-fae3e243731d

[17]https://careerfoundry.com/en/blog/data-analytics/what-is-a-decision-tree/#:~:text=Decision%20trees%20are%20extremely%20useful,%2C%20data%20classification%2C%20and%20regression.