**ANDROID MALWARE DETECTION**

**GROUP : XX**

**1.-------------------**

**2.-----------------**

**3.-----------------**

**ABSTARCT**

The rapid increase in the usage and the development of the android applications, which has been accompanied by a notable increase in the malware, which is going to have serious security threats to users and devices alike. This project focuses on developing and implementing machine learning models to identify and classify Android malware based on system call frequency and binder interaction data. The dataset that we are going to use has the features that are being extracted from the system and the binder call frequencies, which are some of the indicative of various types of malware behaviors.

We are including several machine learning algorithms, which involves Naive Bayes, Random Forest, Logistic Regression, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN). Where each model is subjected to train and tested with the dataset we have, the dataset is a huge one which has 11,598 samples with 470 features, representing different system interactions that are typical of benign and malicious apps. Then after the training and testing they are being evaluated based on the performance metrics which includes accuracy, precision, recall, F1-score, and ROC-AUC values to determine their effectiveness in malware classification.

As we have more features in the dataset, it is required step to select the features that are actually contributing to our projects problem statement to achieve that we need to Feature selection and dimensionality reduction were performed using ANOVA F-test to identify the most significant features contributing to malware detection. The selected features were further processed using a standard scaler to normalize the dataset, which in return ensuring optimal performance of our models we chosen. As we also involved the implementation of the ensemble methods and after the implementation they have outperformed all the other models which are trained and tested earlier. They outperformed in terms of handling the complexity and variety of the dataset. This project not only serves as the advancements in the field of cybersecurity by providing robust models for detecting Android malware but also contributes to the development of safer mobile environments.

**INTRODUCTION**

In this era of rapidly evolving and increasing mobile technology, in that particularly android has emerged as a most used domianant operating system which is powering billions of devices all over the world. So these wide spread made the android as a greatest source to target for the malicious entities, which lead to an increase in the malware that exploits the openness and diversity of the Android ecosystem. As we speak about the android malware it is not only possessing threats to user privacy and data security but also impacts device functionality and network integrity. As these threats are kept increasing there is a need for the effective and efficient detection systems which can easily adapted to the strategies and the approaches employed by the malware developers.

The traditional antivirus methods or the softwares are based on the signature detection and they also struggle to match with the rapid evolution of the new android malwares. These methods often fail to detect new or obfuscated malware until after an attack has occurred, and also there is a critical need for the approaches that are more dynamic and predictive which helps in the securing of android devices against the malicious software. Machine learning offers promising solutions by enabling the development of predictive models that can detect potential threats based on behavioral patterns rather than relying solely on known signatures. This project illustrates the addressing of the challenge of Android malware detection by implementing and comparing several machine learning algorithms that analyze application behavior through system calls and binder interactions.

**MOTIVATION**

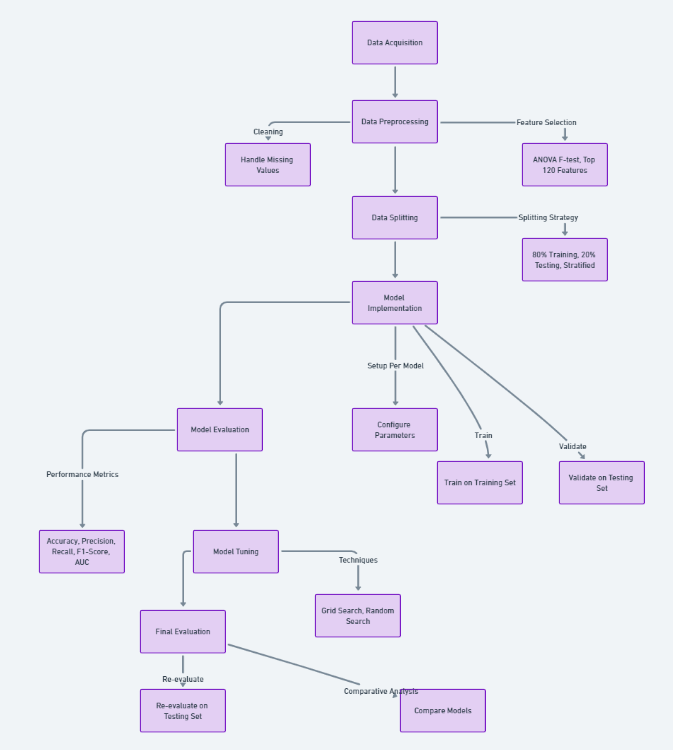
The widespread use of smartphones and the growing usage of the mobile applications for daily tasks such as social networking, entertainment, and banking have made a major impact on the nature of the digital world today. Millions of apps can be downloaded for Android from the app store, the most popular mobile operating system in the world, that satisfy a variety of user demands. But this ease is also accompanied by increased security threats, which is particularly because of the open architecture of the Android operating system and its vast app store.

Although Android is a popular platform among developers due to its open-source architecture and market dominance, it is also vulnerable to a number of security issues. By taking advantage of these weaknesses attackers can produce malware that can steal confidential data, cause financial losses, and even add machines to automated systems. Android malware cases have doubled annually, according to a research from the Internet Security Threat Center, highlighting the ever-increasing threat landscape.

As the old traditional methods that follows the signature based detection are however fundamentally reactive but the thing is that they need prior knowledge of the malware signatures, which makes them difficult to detect the malwares in zero-day exploits—new, previously unknown threats. These limitations of the traditional methods highlights the need for the more sophisticated approach that uses machine learning models which helps in the prediction of the malware. Machine learning (ML) presents a powerful alternative by leveraging patterns and anomalies in data to identify malicious behavior. The main objective is to enhance the security to the android devices against the growing rate of different threats. This project aims to develop ML models that cn adapt to the new malware strategies more effectively than traditional antivirus software.

**WORKFLOW**

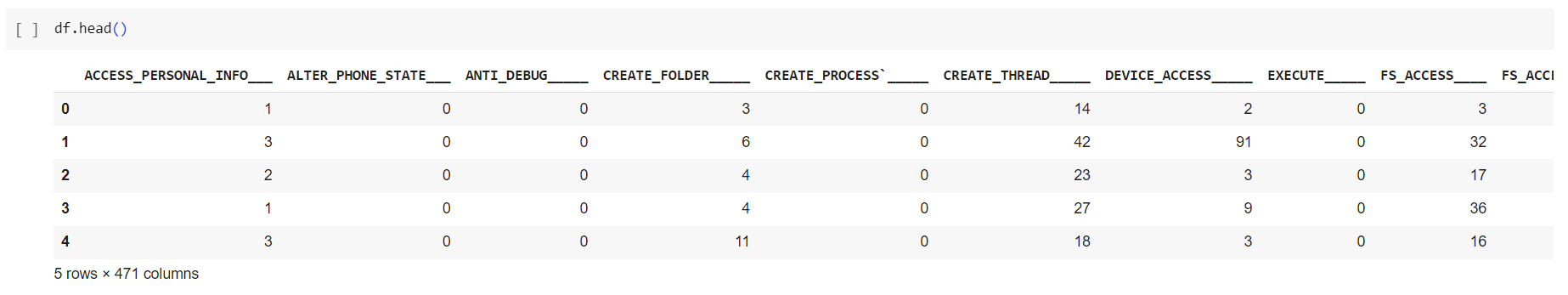
The below is the overall general workflow of this project.



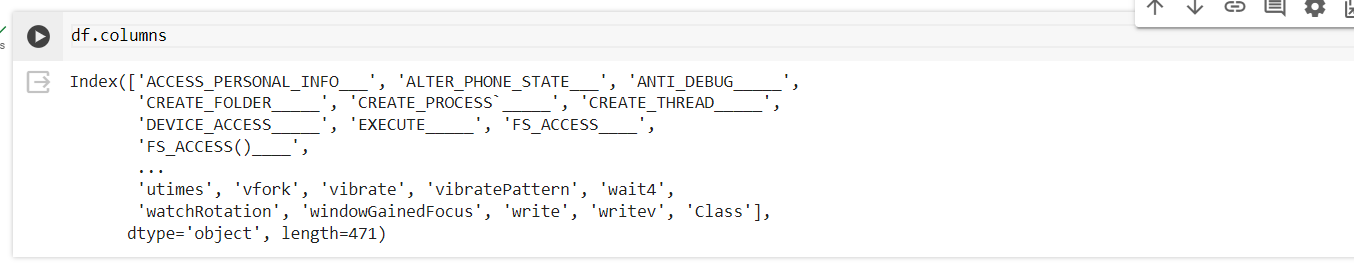
**EDA AND DATASET**

The dataset that we are using in this project has derived from the behaviour of various android applications. The data is specifically engineered to capture the frequency of system calls and binder interactions, which are critical in the operation of Android applications. These interactions are often exploited by malware, making them valuable indicators for detection purposes. The dataset has the 11,598 samples of where it each of them reperesents a single android application. Now as we speak about the features present in the dataset there are 470 features which are very vast in nature, which are specifically focused on the system and binder call frequencies. These features are numerical and represent counts of specific behaviors observed in the application.

The below figure depicts some of the rows present in the dataset:



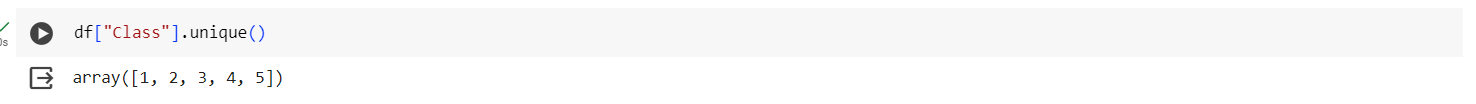
The below figure demonstrates the number of columns in the dataset , in machine learning and the project point of view, they can be stated as number of fetuares are displayed.



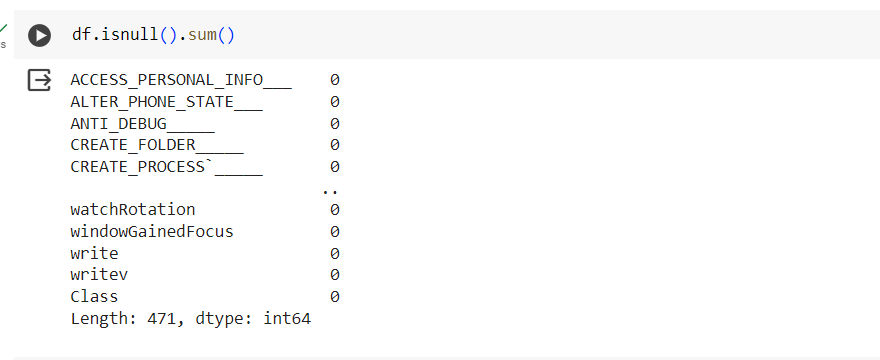
The target variable in the dataset is the class column, where it categorizes the applications into different types of malwares depending on the features that respective application possess. This column is crucial for supervised learning tasks where the goal is to predict the category of unseen applications based on the learned patterns.

The above is the summary of the dataset, below I will be providing some of the preprocessing tasks that being made to make an efficient dataset that can feed to the machine learning models so that the overall performance of the model on the test set will appear more .

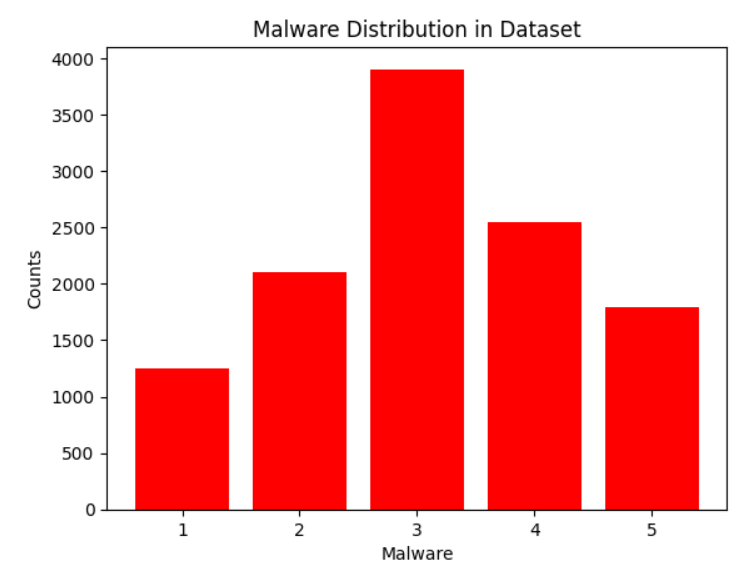
The below image depicts the number of unique classes present in the target variables:



We have also performed some initial checks to check whether there are any null values and the duplicates to remove the data redundancy and making sure the model trains on well defined or designed dataset



Now to gain more insights on the dataset we have visualized the malware distribution in the dataset based on the class it involved, the below is the visualization we have generated as part of it to view the malware distribution.



After the above stages which involved the data cleaning and gaining of insights on the data, now its time to work on the preprocessing in terms of feature selection as we already aware of the fact that we have many features in the dataset, which all of them are not required to train the model in predicting the malware in android. So we are using :

**Statistical Tests :** Here In this we are using ANOVA f-Test, which is used to identify the most significant features that have the strongest relationship with the target variable. This test helped in reducing the dimensionality of the dataset by selecting the top 120 features, thus focusing on the most relevant features for malware detection.

**variance thresholding :** Also we have applied a variance thresholding filter , which is applied to remove features with low variance that do not contribute significantly to the model's decision-making process.

**Data Transformation :** Now the selected features are made to standardize which is to make sure they will contribute equally to the training of the model, which try to remove the bias , due to the scale of the features. The StandardScaler was used to transform the data so that each feature had a mean of zero and a standard deviation of one.

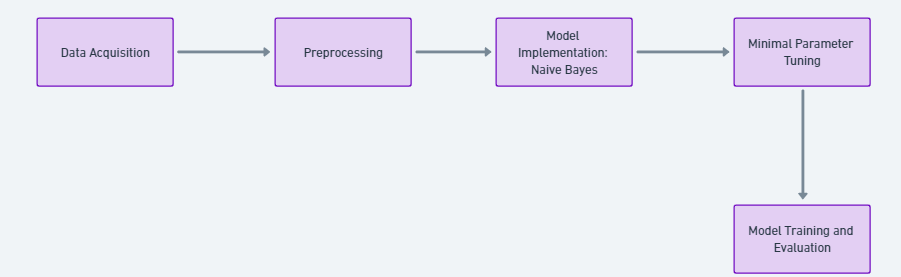
Now at last before feeding the pre-processed data to the model, we need to split the dataset into training and test sets, with which we have chose 80% of the data used for training the models and 20% reserved for testing. This split was done using stratified sampling to maintain the same proportion of each class in both training and test sets, ensuring that the models are tested against distribution of classes.

**MODELS**

Below are the models that we have trained with the preprocessed dataset and evaluated them using the test set, the distinct models we used are namely : Naïve Bayes classifier, Random forest classifier, Logistic regression, Support vector machines, K-Nearest Neighbors(KNN).

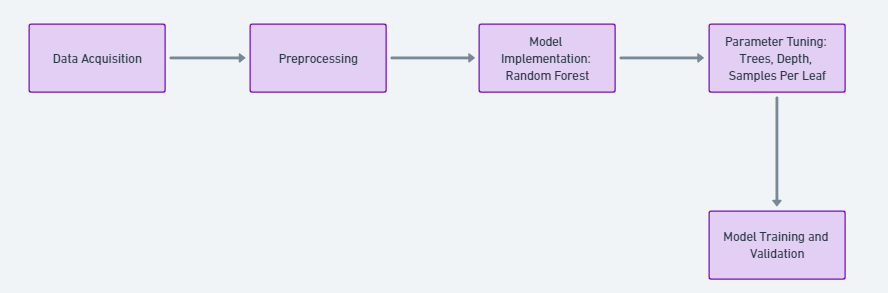
1. **Naïve Bayes Classifier:** Generally the Naïve Bayes is being chosen for its simplicity and efficiency . The working principle behind this model is the conditional probability as it being stated as the Bayes Theorem and here assuming features are independent given the class label.

* **Data Pre-Processing :** Here in this stage features were standardized to make sure they are on the same scale, which is not required in the case of Naïve Bayes classifier, but it will help in maintaining the consistency in the preprocessing steps in all the models
* **Model Training :** Now the model will gets trained on the system calls and binder calls as the feature variables and the target variable is as we discussed the malware class.
* **Hyperparameter tuning :** Generally the Naïve Bayes model requires minimal tuning. however, the prior probabilities of each class were set based on the distribution observed in the training data to handle class imbalance.
* **Model Evaluation :** Now the model will be evaluated on the test set, which is having performance metrics such as accuracy, precision, recall, and F1-score calculated to assess its performance.



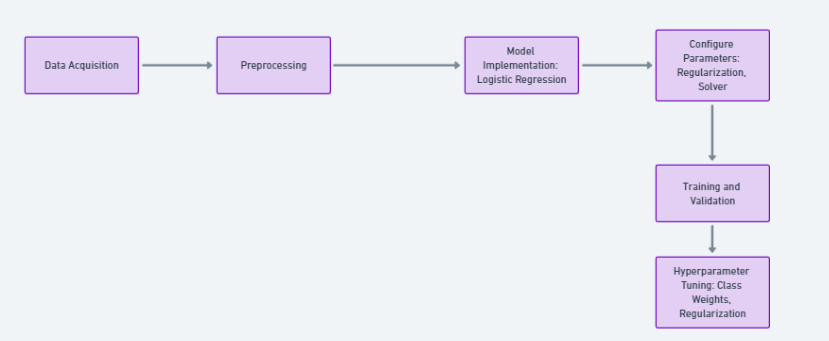
1. **Random Forest Classifier :** Generally this model is an ensemble learning method, which is based on the aggregation of the predictions of the multiple decision trees to improve the predictive accuracy and control over-fitting.

* **Data Preprocessing :** Here the data processing is done similar to above model and it is similar for all the models.
* **Model Training :** Here in this stage a large number of decision trees were constructed, each trained on a random subset of features and samples. Then after all the earlier steps the predictions are done based on the averaging the predictions from all the trees.
* **Parameter Tuning :** The parameters like the number of trees (n\_estimators), the maximum depth of trees, and the minimum number of samples required to split a node were tuned using cross-validation techniques to find the optimal settings.
* **Model Evaluation :** Now the model will be evaluated on the test set, which is having performance metrics such as accuracy, precision, recall, and F1-score calculated to assess its performance



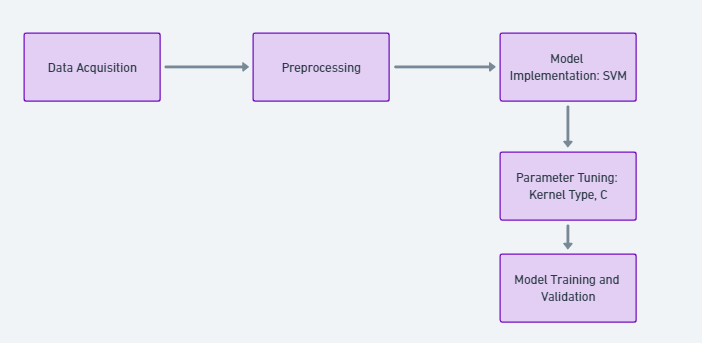
1. **Logistic Regression :** In this model , it is used to provide the probabilistic outputs and it also models the probability of classes with the help of logistic function.

* **Data Preprocessing :** Standardization was applied to the dataset to optimize the performance of the logistic regression model.
* **Model Training :** The approach, which is especially helpful for binary classification issues extended to multiclass classification via the one-vs-rest technique, employed a logistic function to predict the chance that provided inputs correspond to the default class.
* **Parameter Tuning :** In this we will be tuning the Regularization strength and the type of solver used for optimization , which helps in the enhancement of the model performance and also allows to prevent the overfitting.



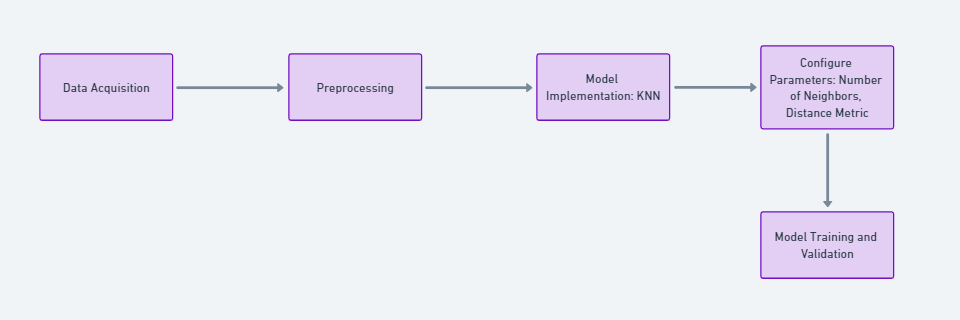
1. **Support Vector Machine (SVM) :** We have selected this model, as it is effective in the high dimensional space, which makes it ideal for the datasets that are having large number of features like in our dataset which has system calls and binder calls.

* **Data preprocessing :** It is same for all the models we chose.
* **Model Training :** The SVM was trained using a linear kernel initially to maintain computational efficiency, with the soft margin of classification tuned via the C parameter.
* **Parameter Tuning :** The regularization parameter (C) was carefully tuned, which helps in balancing the trade-off between achieving a low training error and also maintaining the ability to generalize to unseen data as well.
* **Model Evaluation :** The performance was carefully assessed using accuracy, precision, recall, and F1-scores, along with ROC curves which will help in understand the trade-offs between true positive rates and false positive rates.



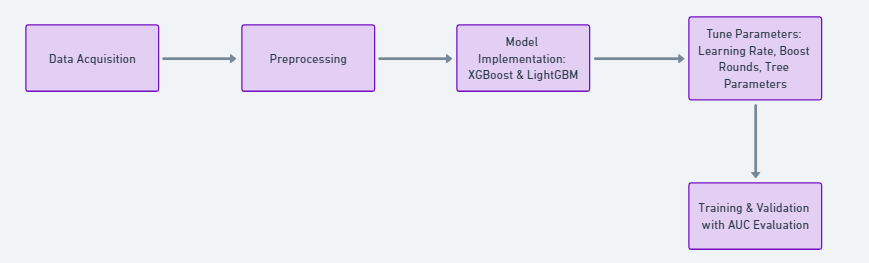
1. **K-Nearest Neighbors (KNN) :** KNN is generally know for it simplicity and the effectiveness it gives to the tasks that involves the classification, which allows to capture the structure of the data, which is generally advantage for the detection of patterns from the types of malware.

* **Data Preprocessing :** As we discussed it is just the same with all the other models, and it is very crucial in the KNN because of the sensitivity it is having with the range of datapoints.
* **Model Tuning :** The number of neighbors here is the hyperparameter in the KNN, which is tuned to get the best fit model out of it.
* **Model Evaluation :** This model is evaluated to just make sure it is capturing the right patterns and also making sure the model doesn’t overfit with the model. Here as always we use standard performance metrics.

****

1. **XGBoost and LightBGM :**

* This models are generally the ensemble methods, where this models gave us the high performance metrics.



The above descriptions of all the models used generally gives the idea of how the models are implemented as part of the android malware detection.

**EVALUATION**

In this project as we shown the implementation of different machine learning models, the evaluation of those models for Android malware detection highlights significant variances in performance across different metrics including accuracy, precision, recall, and F1-score. Here we will be outlining the performance of all the models:

**Logistic Regression**

Accuracy: 80.78%

Precision: 80.79%

Recall: 80.78%

F1-Score: 80.33%

**SVM (Support Vector Machine)**

Accuracy: 82.63%

Precision: 82.64%

Recall: 82.63%

F1-Score: 82.34%

**Random Forest**

Accuracy: 94.22%

Precision: 94.31%

Recall: 94.22%

F1-Score: 94.21%

**KNN (K-Nearest Neighbors)**

Accuracy: 90.09%

Precision: 90.45%

Recall: 90.09%

F1-Score: 90.11%

**Naive Bayes**

Accuracy: 58.92%

Precision: 69.36%

Recall: 58.92%

F1-Score: 54.62%

Additionally, two advanced ensemble models were tested:

**XGBoost**

Accuracy: 95.30%

Precision: 95.32%

Recall: 95.30%

F1-Score: 95.29%

AUC: 99.65%

**LightGBM**

Accuracy: 95.17%

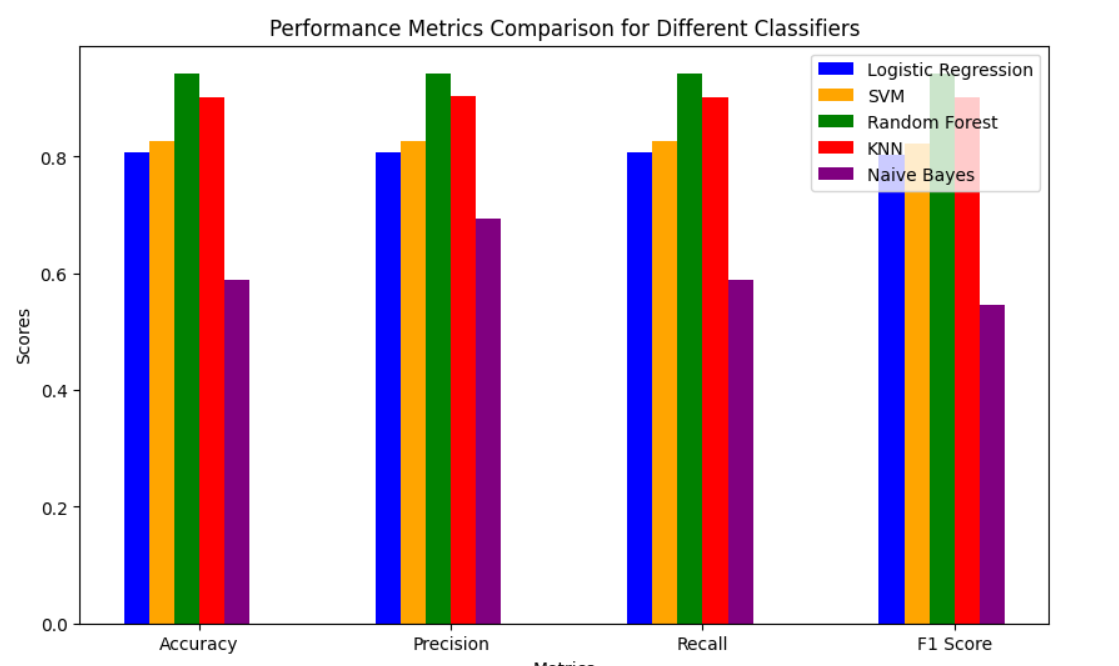
Precision: 95.21%

Recall: 95.17%

F1-Score: 95.17%

AUC: 99.66%

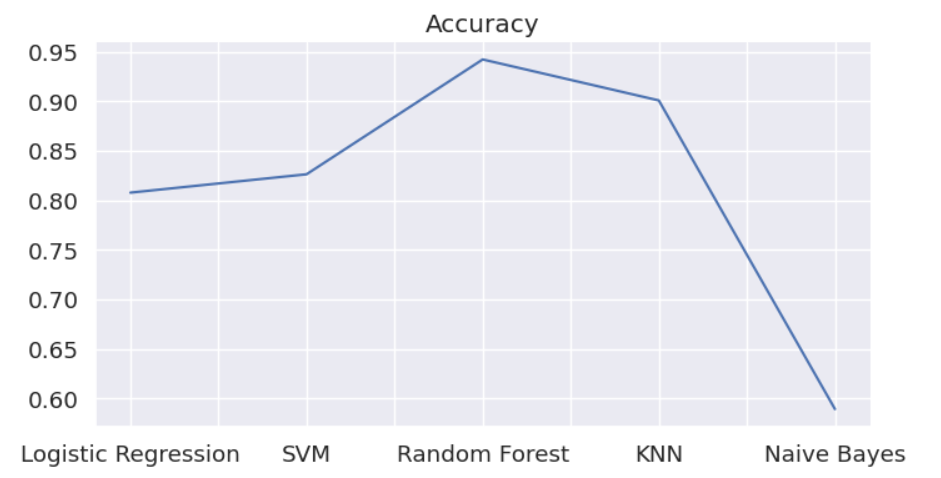
In the below image, we can see that the graph is plotted against the scores that we got as part of the model performance for different parameters and different models visualized below.

****

The below is the confusion matrix for the model we have implemented as part of this project.

****

The below visualization clearly depicts the performance metric , In particular with the accuracy of the model where it clearly shows that random forest has the highest accuracy among all the other models, except with the XGBoost and LightGBM. Even as we speak about the technical aspects, all the three models that gained high performance are part of ensemble methods. So by the inclusion of ensemble methods we have gained high accuracy and as an overall project, the performance of the ensemble methods are high.



So all this provided charts and the visualizations clearly depicts the performance metrics of all the models used. The bar chart shows a comparative analysis across all metrics, clearly indicating the Random Forest, XGBoost, and LightGBM models as the top performers with high scores across accuracy, precision, recall, and F1-score. The heatmap complements this data by offering a detailed view of each model's scores. Which is giving us more detailed view of the models efficiencies in all the aspects of the android malware detection.

The assessment highlights how well ensemble approaches—in specific, Random Forest, XGBoost, and LightGBM—handle the complex nature of Android malware detection. These models showed durability in managing the inconsistent and diverse information characteristic of real-world situations, in addition to offering better performance signs. Even though it failed as well, the Naive Bayes model provided insightful information on the restrictions and difficulties associated with applying probabilistic models to severely unbalanced datasets.

**CONCLUSION**

To conclude, this project aims to develop a robust android malware detection system from different types of machine learning models that have demostrated a different findings as it highlights the effectiveness of advanced analytical techniques by improving measures that are related to cybersecurity. The methods to perform together are, particularly Random Forest, XGBoost, and LightGBM, while they are known as top performers, high accuracy exhibiting, precision, recall, and F1 scores. To handle the ability of time complexity and feature a rich type of nature for android malware dataset which was demonstrated clearly and effectively. The two SVM and KNN have also showed a way more than expected performance by proving the capabilities by handling a high dimensional data as they were somewhat outperformed by ensembling the approaches. Whereas using the Naive Bayes Model, as it is useful for their speed and simplicity and is dealing with the hard featured interactions and the data imbalancement. While using the ANOVA F-test for selecting the features which was more important to .boost the performance of the model by mainly focusing on the most useful features in the list, and by avoiding over fitting models and improving the generalization of the models different abilities.

From the findings, we found out that it is important for those who are in cybersecurity domain, especially those which contain security of the mobile. The capability to identify and categorize different malware of the android successfully enables the advanced security measures, by reducing the chance of malware infection before it causes any harm to the device. In addition, the insights that are provided from the significance of feature analysis that can also assist different security specialists while focusing on the most important parts of application behavior, hence by improving current security protocols and methods of the security.

**FUTURE WORK**

Now as we move on to discuss about the future work, there is chance to explore towards the usage of Deep learning models which can still get better results, in which they are able to capture the dependencies in system call sequences more effectively. Even Expanding the dataset to include more examples of the latest malware variants could improve the models' robustness and adaptability to new threats. Also, Implementing these systems in a real time detection system can also be explored to work on it, potentially integrating them into existing Android security frameworks to provide dynamic threat detection and response.

**REFERENCES**

1. Demontis, A., M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. 2017. Yes, machine learning can be more secure! a case study on android malware detection. IEEE Transactions on Dependable & Secure Computing PP (99):1–1.
2. Engelen, and Hoos. 2020. A survey on semi-supervised learning. Machine Learning 109:02.Faiz, M. F. I., Hussain, M. A., and Marchang N. (2020). Android malware detection using multi-stage classification models. In Conference on Complex, Intelligent, and Software Intensive Systems Lodz, PL, 244–54. Springer
3. Fatima, A., Maurya, R., M. S. Dutta, Burget, R., and Masek J. (2019). Android malware detection using genetic algorithm based optimized feature selection and machine learning. 2019 42nd International Conference on Telecommunications and Signal Processing, Budapest, Hungary, TSP 2019, 220–23.
4. Firdausi, I., Lim, C., Erwin A., and Nugroho A. S. (2010). Analysis of machine learning techniques used in behavior-based malware detection. In Proceedings - 2010 2nd International Conference on Advances in Computing, Control and Telecommunication Technologies, ACT 2010, Washington, DC, 201–03
5. Hasegawa, C. Iyatomi, H. (2018). One-dimensional convolutional neural networks for Android malware detection. In Proceedings - 2018 IEEE 14th International Colloquium on Signal Processing and its Application Penang, Malaysia, CSPA 2018