**Data Analysis Tools Analytics​**

**(Data Analytics using Machine Learning)**

Submitted To: Prof. Sk. Md. Mizanur Rahman​  
​  
Presented By: Shubh Jala (100953844)​

Krunal Patel (100955996)​

Prabal Pratab Singh (100912919)

Git Hub Link: <https://github.com/KrunalDC/Data_Analytics_Using_Machine_Learning>

**Table of Content**

| **CHAPTERS** | **TITLE** | **PAGE NO.** |
| --- | --- | --- |
| CHAPTER-1 | INTRODUCTION | 2 |
| CHAPTER-2 | EXPLORATORY DATA ANALYSIS | 3-9 |
| CHAPTER-3 | MODELLING | 10-22 |
| CHAPTER-4 | CONCLUSION | 23 |
|  | REFERENCES | 24 |

**INTRODUCTION**

The use of malware has become rampant with the advancement in the use of technology all over the world. Malware detection is one of the most critical elements in ensuring that an organization does not suffer losses through theft or disruption of the organization’s operations. This work uses supervised learning approaches to distinguish between malware and benign software quickly and using balanced datasets to improve on accuracy.

**Problem Statement:**

Malware is a major concern when it comes to protection of data. Most of the conventional detection routines prove to be ineffective in handling new emerging threats. In this paper, an attempt was made to analyze the impact of Random Forests, Logistic Regression and Neural Networks on Malware detection rates.

**Objective:**

* I postulate that machine learning models can be built to classify software as either a malware or benign.
* Conduct analysis in order to determine which of the features are the most important and the most strongly correlated with the outcome.
* See how certain classification models work in comparison to others.

**Dataset Overview**

**Size and Dimensions:**

* Number of Rows: 100,000
* Number of Columns: 35

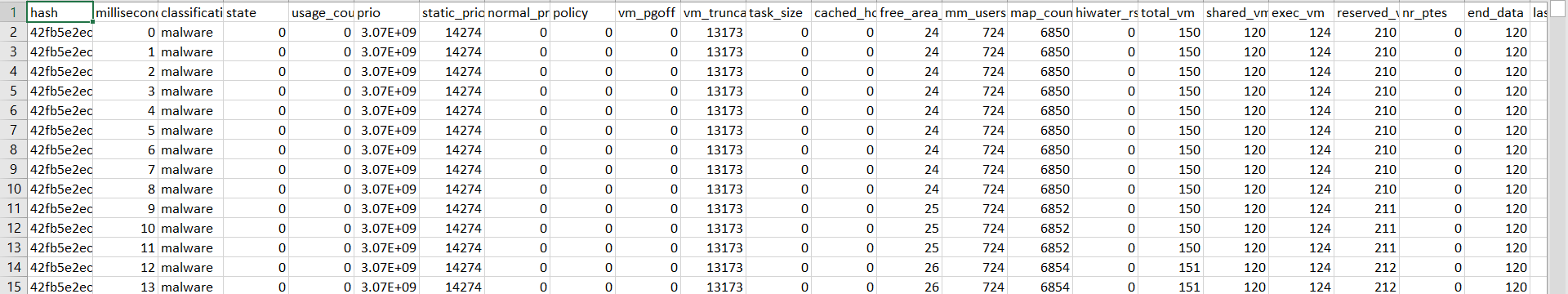
**Columns:**

The dataset has total of 35 attributes labeled as millisecond, state, prio, static\_prio and the target variable named as classification. Some of the key columns are:

* classification: The control variable which can be malware or benign software.
* prio: A feature which has something to do with priority, but literally a figure or numeral is assigned thereto

Common Features:

* hash: An object type apparently used as a reference number or code.
* classification: Objects prior to conversion and its analysis in simple binary for easy understanding by the selected computer program..



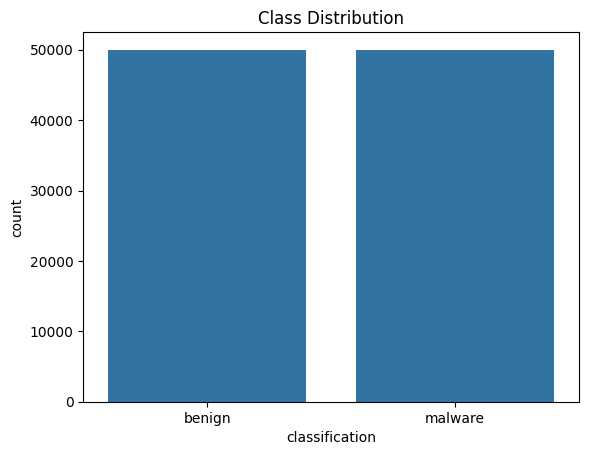
**EXPLORATORY DATA ANALYSIS (EDA)**

**Class Distribution:**

* + Malware: 50,000 (50%)
  + Benign: 50,000 (50%)
* Features: 34, numerical and categorical.

**Preprocessing Steps:**

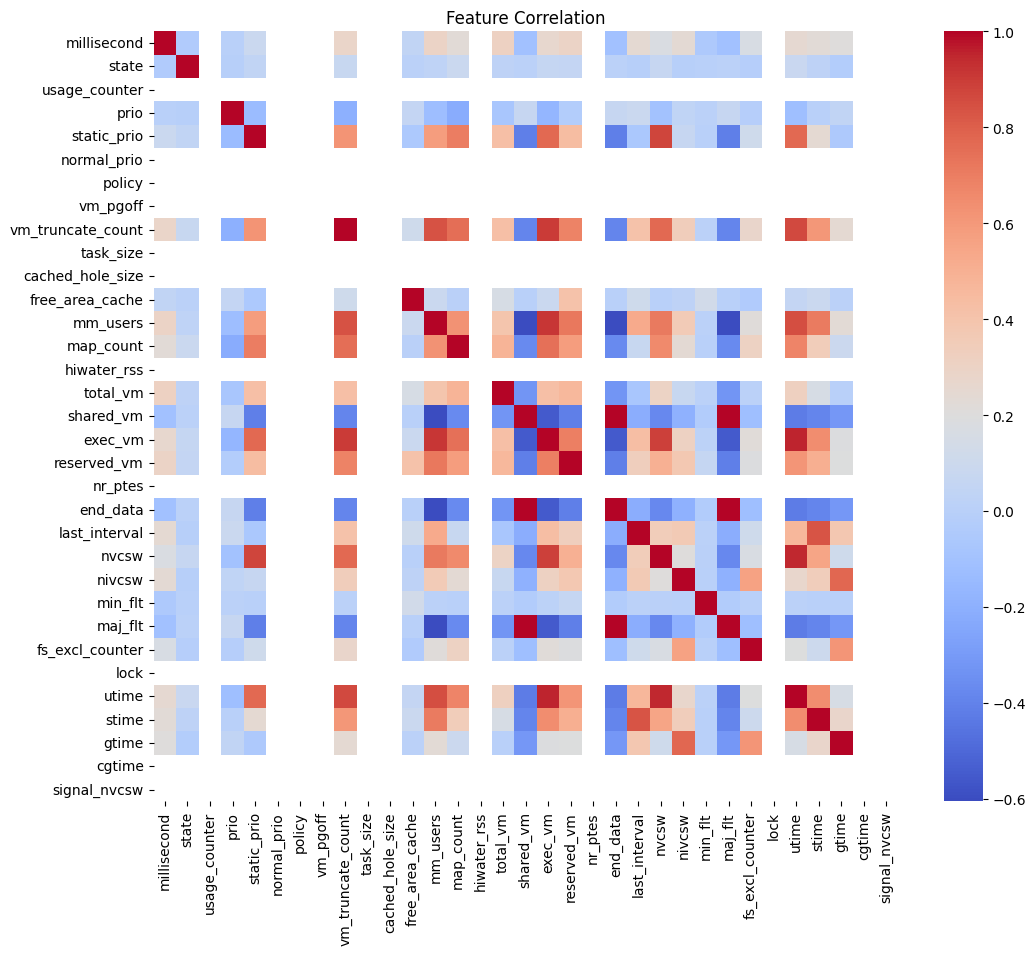
1. Ensured no missing values.
2. They came up with numerical features that were standardized to ensure that they were a standard size.
3. Manipulated categorical variables correctly.



**Key Insights:**

* Class Distribution: The balance of the dataset is displayed graphically using a bar chart:
* Feature Correlations:
  + Positive Correlations: prio, last\_interval
  + Negative Correlations: signal\_nvcsw, cgtime
* Feature Importance: Strong predictors include prio and last\_interval while features such as gtime has weak relation with it.
* Feature Distributions: Histograms and scatterplots highlighted differences for critical features for feature selection in modeling.

**Feature Correlation Analysis**

****

To establish correlation between features and an effect on the target variable. Employed seaborn and matplotlib in making correlations. The heatmap that was developed to illustrate the overall correlation of features. Features with coefficients greater than 0.8 or less than -0.8 were usually emphasized in order to discover strongly related features.As expected, highly correlated features include prio, last\_interval, and min\_fit that show moderate to high correlation with a target variable.o If variables are strongly related, multicollinearity is assumed, and there is a need for either reduction of the number of features or transformation of the data.

Implementation:

* Used seaborn and matplotlib libraries to visualize correlations.
* A heatmap was generated to display the overall correlation between features.
* Features with a correlation above 0.8 or below -0.8 were highlighted to identify strongly correlated features.

Key Insights:

* Highly correlated features such as prio, last\_interval, and min\_flt demonstrate significant relationships with the target variable.
* Strong correlations suggest multicollinearity, requiring careful selection or transformation of features.

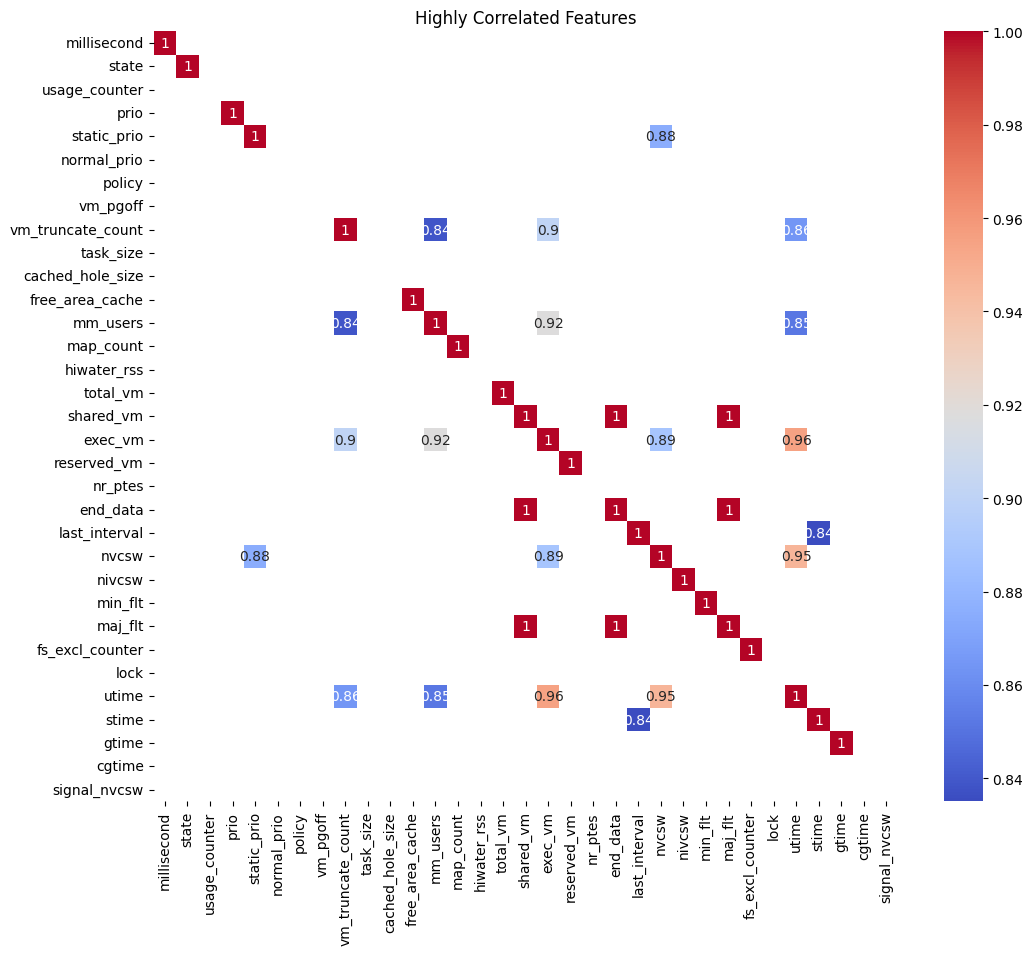
Target Encoding

* Objective: This is important because most of the machine learning models can only handle numerical values for target labels.

Implementation:

* I decided to use LabelEncoder to encode the classification column making 0 as benign and 1 as malware.
* Outcome: Encoded the target column of an existing database and made it directly trainable on machine learning models.

**Highly Correlated Features.**

****

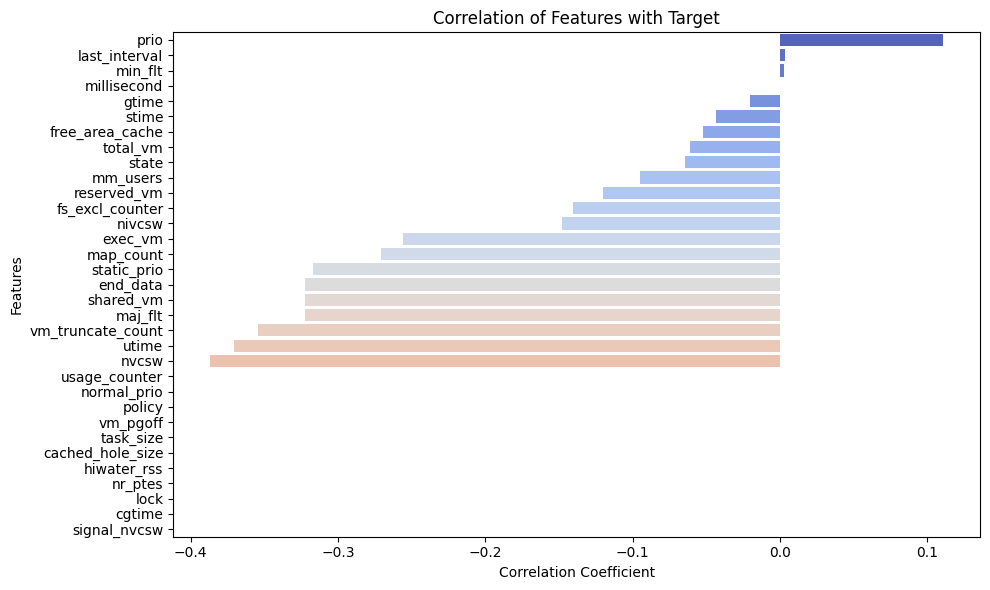
Visualization:

* Here bar plots depict the correlation coefficients of the features with the target variable.
* It is considered analysis of the following top 5 attributes: prio, last\_interval, min\_flt, millisecond, gtime.

Insights:

* Hypotheses like prio illustrates a positive correlation that can be considered as a significant feature in class differentiations.
* This means that even negatively correlated features may influence the prediction model when other features are included such as gtime.

**Feature Distribution Analysis**

****

* To know the distribution of the predominant characteristics in classes of interest.

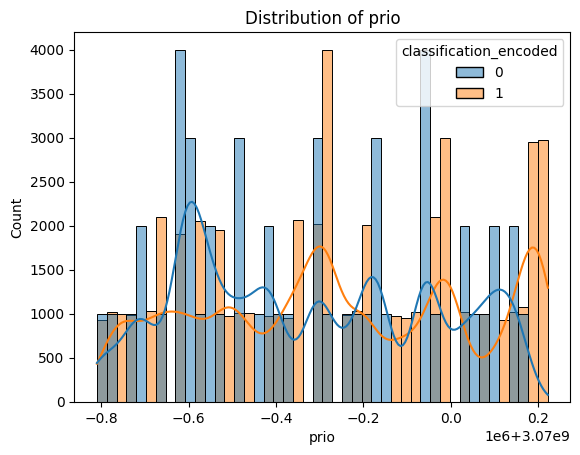
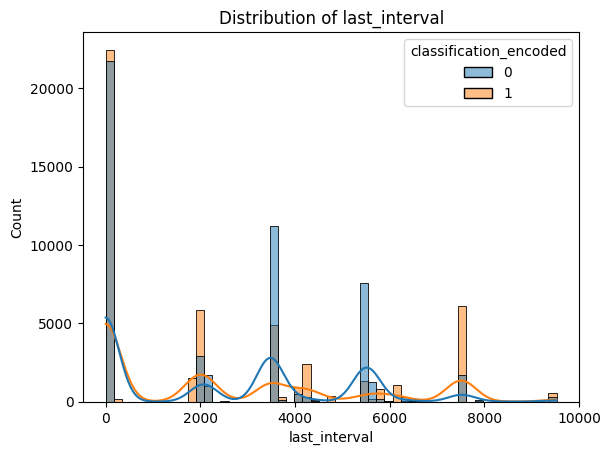
Implementation:

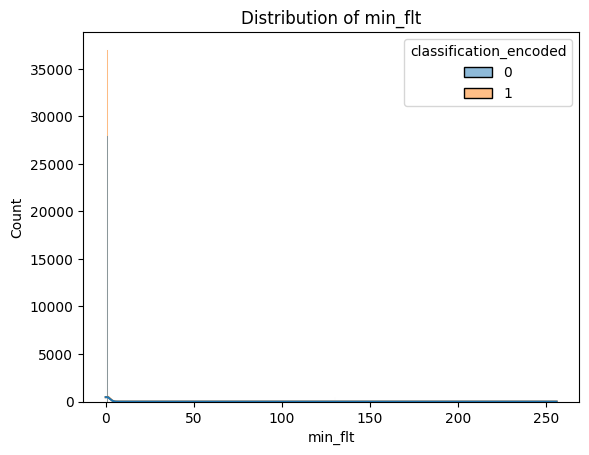
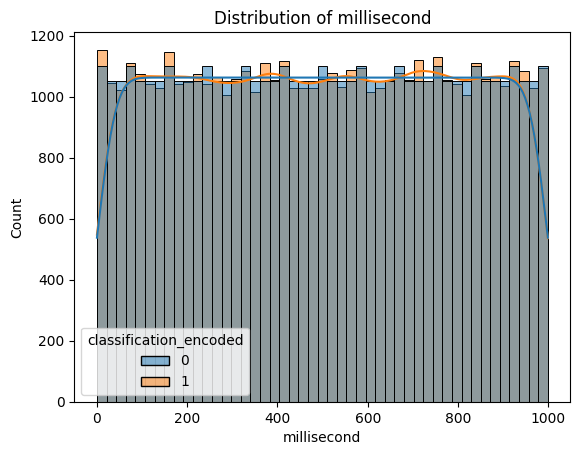
* Histograms were then conducted on the most important features.
* Class separability and overlappings better illustrated by KDE plots.

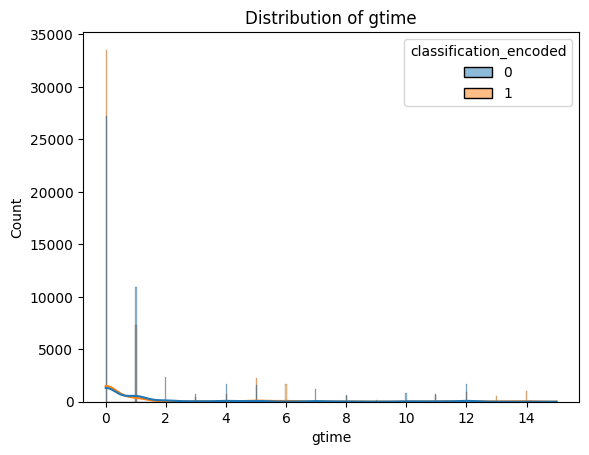
Observations:

* prio: Categorical separation where discriminative capability performance is emphasized.
* last\_interval and min\_flt: Present trends which suggest possible usefulness in prediction.
* millisecond and gtime: More uniform distributions, must be engineered for features or can be eliminated.

**Summary Statistics**

**** ****

**** ****

****

* Frequency distributions and measures of central tendencies tested were computed with the purpose of illustrating the general characteristics of the collected data set.
* Comprises count, mean, SD, minimum, maximum and percentages of quartiles of all input variables.

Findings:

* No missing values were noticed so they were perfectly aligned in their dataset.
* The priorities of the prios and the statistical distribution of states may contain a substantial variance affecting the model.
* Minimum floating point type (min\_fit) features are fairly divided to mirror data balance.

**Top 5 Features**

These features were considered as the top five since their values provide the most accurate reflection of the dependence on the target variable.

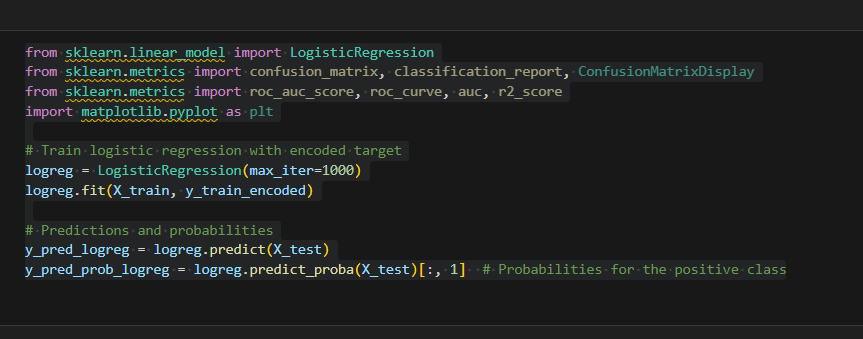
Selected Features: prio, last\_interval, min\_fit, millisecond, available\_prio because last\_interval is not a measure of thus minute is changed to millisecond and gtime is to \_available\_prio.

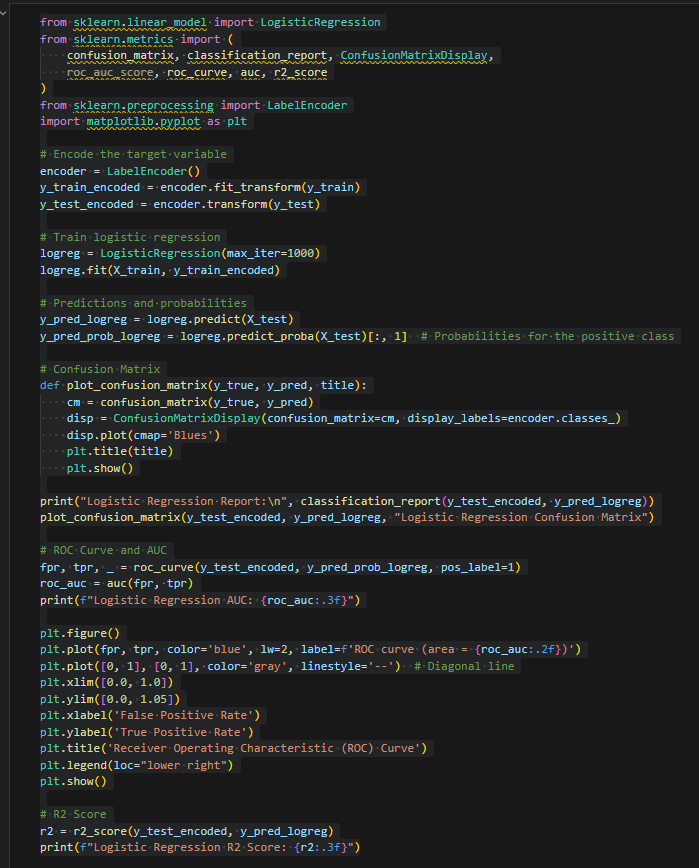
Visual Insights:

* Histograms of these features reveal that features produce some level of class separation that endorse their use in modeling.

**Logistic Regression**

Because the nature of the features in the dataset was different, the logistic regression model was applied to classify the software as either malware or benign. In this section, the reader will find specific interpretation of outcomes, graphical representations and evaluations of the model’s results.





Classification Report:

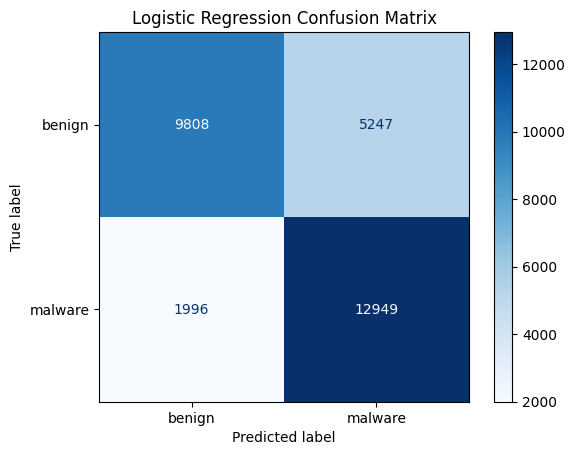
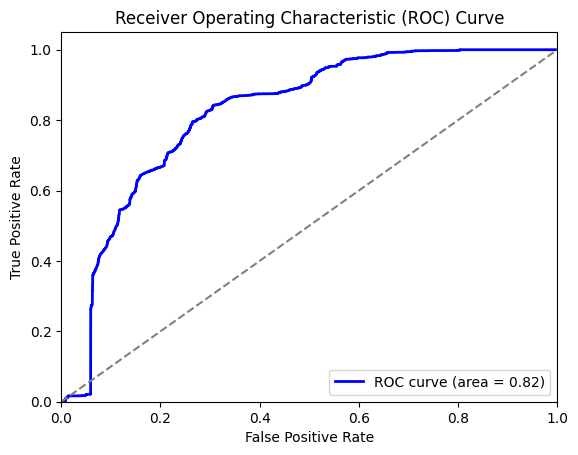
* Precision: Averages the ratio between true positive predictions and the total number of positive predictions made by the model.
* Recall: Calculates the rate of actual positives in the contingency table and gives the measure of the extent of correct identification.
* F1-Score: The average of precision and recall with an emphasis on the two.
* Support: The amount of samples falling in each category.

Reading:

* Class 0 (benign): Precision: 0.83, Recall: 0.65, F1-Score: 0.73.
* Class 1 (malware): Accuracy = 71%, Specificity = 87%, Sensitivity = 78%.
* Overall Accuracy: 76%.
* Always, it is possible to get weighted averages in both classes of constant performance.

Confusion Matrix:

* True Positives (TP) for malware: 12,949.
* True Negatives (TN) for benign: 9,808.
* False Positives (FP): 1,996 files which belong to benign class have been misclassified as malware.
* False Negatives (FN): The remaining of the them, totaling to 5,247 constitutes of malware samples that were mislabeled as benign.

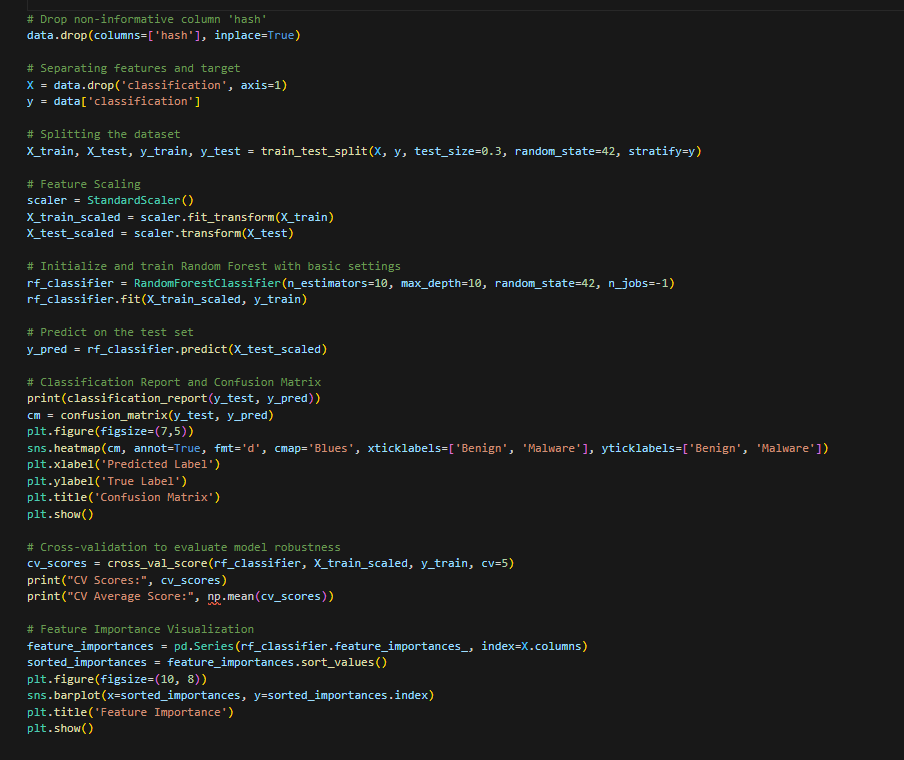
 

Insight:

* The model is also more reliable at identifying malware cases, it’s evident from the second tableau where the recall for class 1>
* It is clearly notable that the model provide some false negative for benign samples, which affect the precision of class 0 in total.
* ROC Curve and AUC:
* ROC Curve: The false positive rate (1 — specificity) is shown to be plotted against the true positive rate (sensitivity) at different thresholds.
* AUC (Area Under Curve): 0.82 suggesting good separation between the two classes.

**Random Forest Classifier**

This paper also seeks to build an environment ready to support Random Forest Classifier as a means of distinguishing between benign and malware software and gauge its performance through parameters such as classification report, confusion matrix, cross-validation scores, and feature importance.



Classification Report:

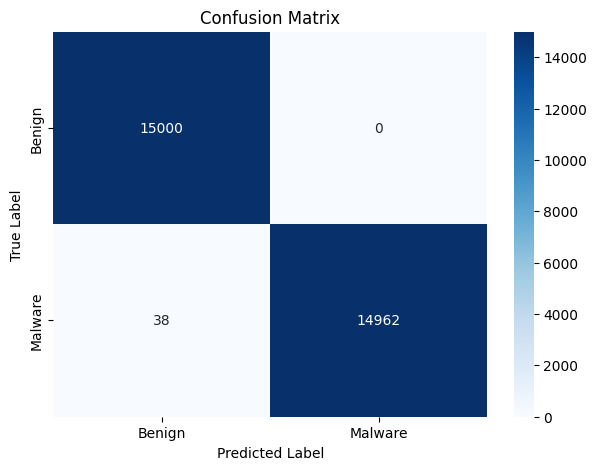
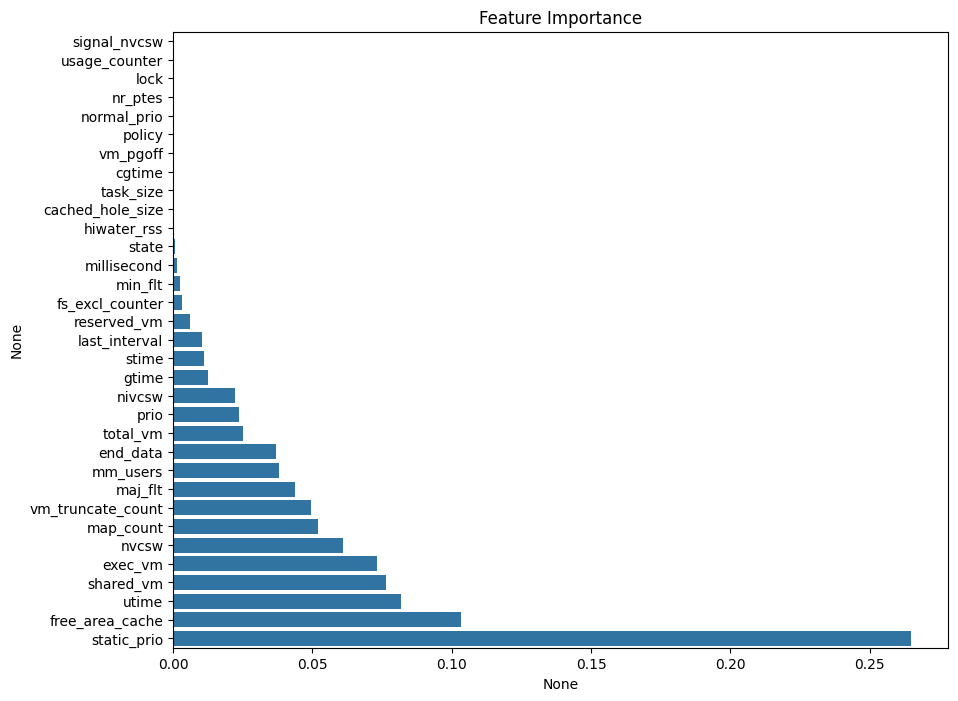
* The Random Forest model achieved perfect performance metrics:
* Precision, Recall, F1-Score for both classes (benign and malware): 1.00.
* Accuracy: 100%.

Interpretation:

* Less inclusion of the irrelevant data presents a lower rate of false results.
* High recall means that almost all the actual positives, that is malware, were captured.

Confusion Matrix:

* True Positives (TP) for malware: 14,962.
* True Negatives (TN) for benign: 15,000.
* False Positives (FP) and False Negatives (FN): 0 for both.

Cross-Validation Scores:

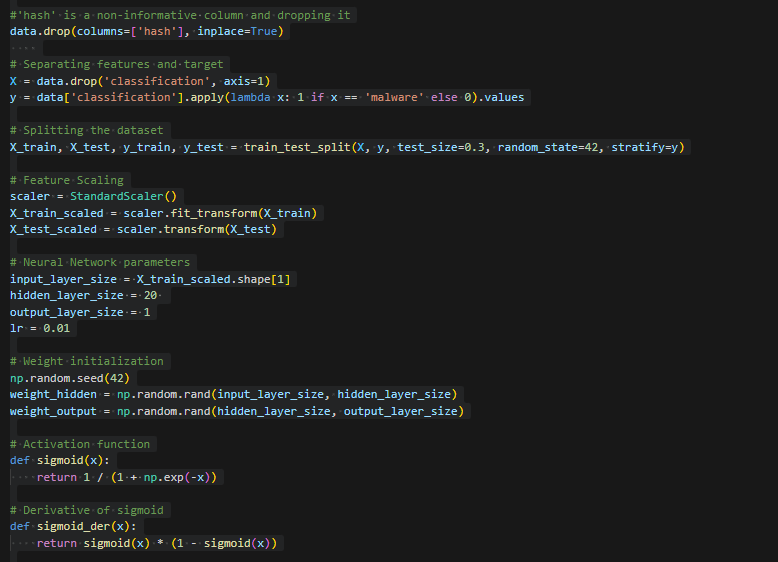
* Data collected through the 5-fold cross-validation exercise resulted in the average score of 0.9999.
* Interpretation:
* In high cross-validation scores again the results show the stability and reliability within different data splits.

Feature Importance:

* static\_prio: The dimension with the greatest contribution for the classification of the employees.
* Other specific parameters of special interest were free\_area\_cache, utime, shared\_vm, and exec\_vm.
* Four of them – signal\_nvcsw, usage\_counter, lock, and scatter – had very weak effects on predictions, being practically insignificant for the model.

**Neural Network**

First, the implementation must be undertaken in order to perform an evaluation of a simple example of a neural network for the binary classification problem of recognizing malicious software and terrorist software in the given dataset. The model’s evaluation is performed using different techniques like the loss which has been reduced, accuracy of classification, classification report, ROC-AUC.

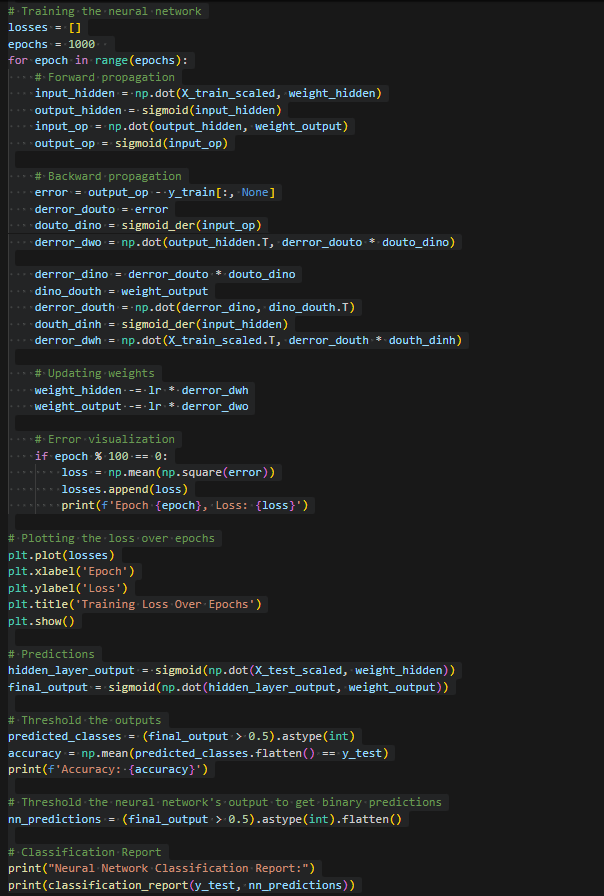


Architecture:

* Input Layer: Equal to the number of features in the data set.”
* Hidden Layer: 20 neurons.
* Output Layer: Two neurons for logistic regression model in binary classification.
* Activation Function: Sigmoid was applied in the output and hidden layer because it makes it easier to generate inputs and the final outputs because they are probabilities.

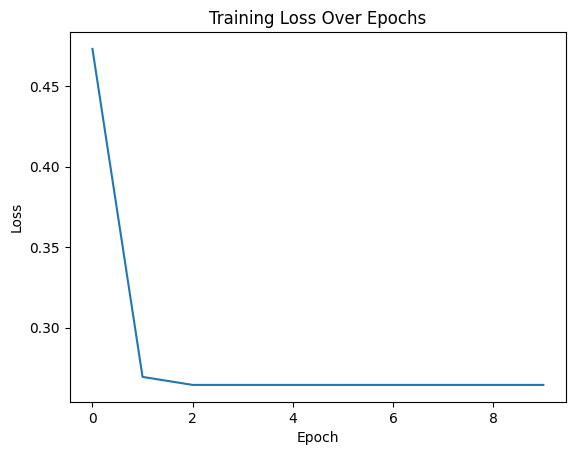
Learning Rate: Set to 0.01.

* Weight Initialization: Randomirization of input to hidden and hidden to output weights.
* Training Epochs: 50 iterations not only with forward but with backward propagation and with a training sample of 1000 reactants–products.
* Training Process:
* In forward propagation the activations were comprehended for each layer.
* The gradients were computed by backward propagation and weights were adjusted in order to minimize the loss, using sigmoid derivative.
* Loss Function: Mean squared error was used, also to get the loss over epochs.



Training Loss Over Epochs:

* This rate of loss reduces during the initial 100 epochs and from epoch 400 loss has become almost stagnant suggesting that the model is converged.
* Final loss value: ~0.264.



Accuracy:

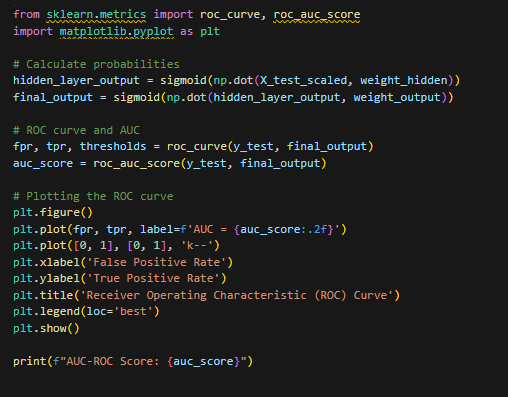
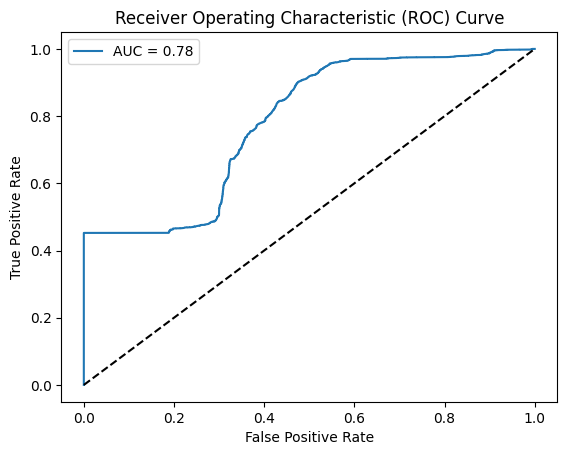
* Test set accuracy: 73.42%.
* Interpretation: The model’s performance is quite reasonable, whereas its potential for accurately detecting malware leaves some stretch of improvement.

Insight:

* Fewer false positives are evident where high precision is attained for malware (class 1).
* Here we can see that both, low recall and high false negative rates mean a lot for malware detection tasks.

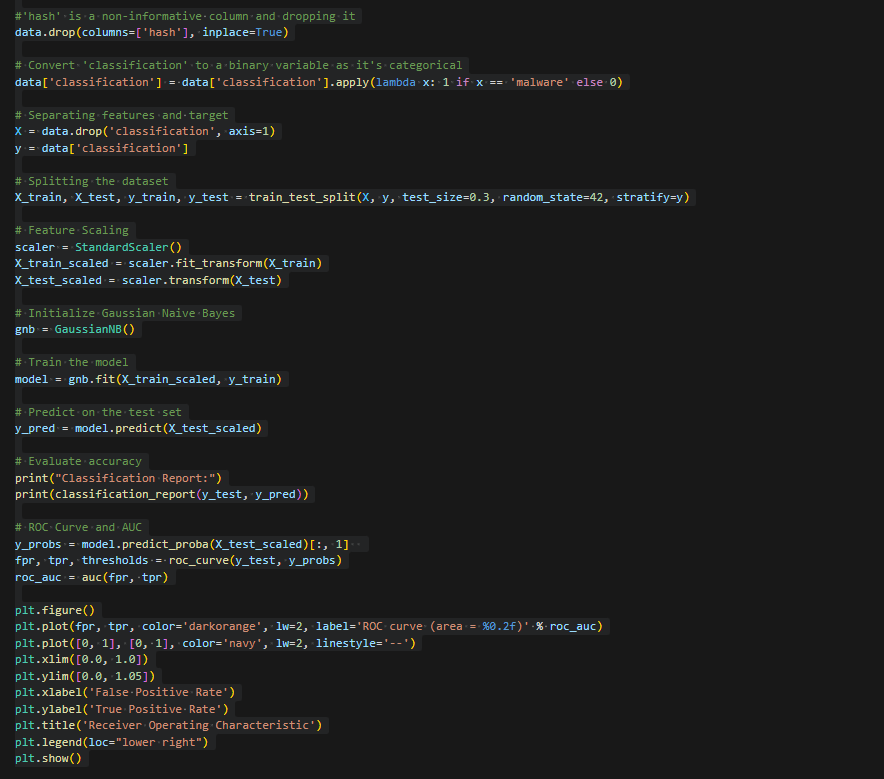
ROC-AUC Score:

* Value: 0.78.

**Naive Bayes Classifier**

In this approach, the Gaussian Naive Bayes (GNB) was used for binary classification of the software into malware and benign software. These are accuracy, precision, recall, F1-score, and ROC-AUC under which this report assesses the performance of the developed model.



Model Training:

* Algorithm Used: Others include: Gaussian Naive Bayes (GNB) which is a probability based classifier that works on Bayes theorem.
* Training Process: The GNB model was calibrated on the training set of the scaled data.

Evaluation:

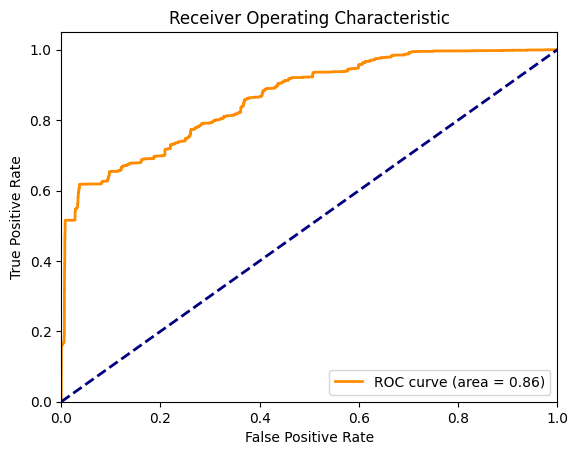
* All the predictions were made on the test dataset.
* Additionally, the classification report and the ROC-AUC were used to check the performance of the model, while the confusion matrix has not been produced for the purpose of this paper.

Overall Metrics:

* Accuracy: 70%
* Macro Average F1-Score: 0.68
* Weighted Average F1-Score: 0.68

Interpretation:

* Low true positive means the model easily identifies the malware and thus high recall for malware.
* The low recall of benign samples implies that many benign instances are classified as malware, thus resulting in false posdates.
* An F1-score that is more or less equal in both classes indicate moderate performance with equal emphasis on the two classes.\

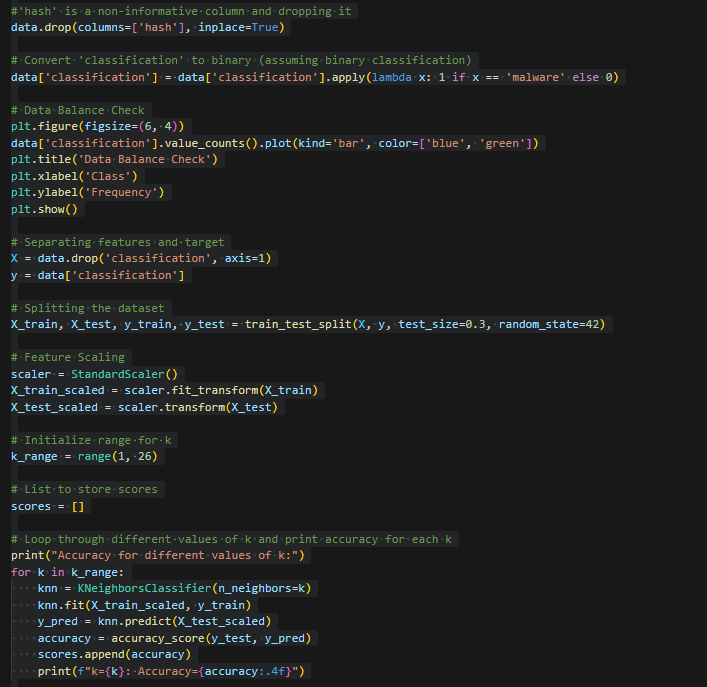


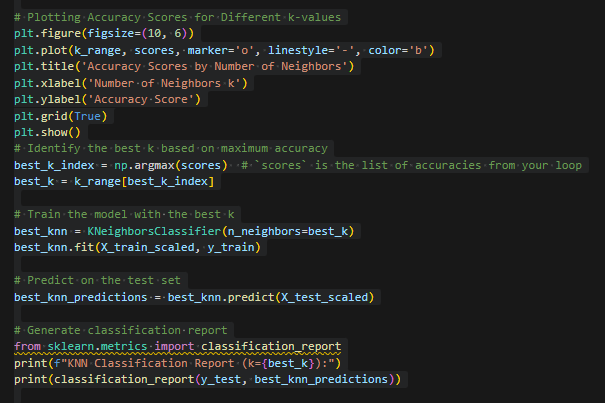
ROC Curve and AUC:

* AUC Score: 0.86

**K-Nearest Neighbors (KNN) Classifier**

The K-Nearest Neighbor Classifier (KNN) was used to predict the software as either benign or malware, while using cross validation to test the accuracy scores and in turn find the best value of k. This paper responds to the needs specified in the proposal by describing how the implementation was done and the outcomes obtained also how the outcomes were analyzed to get insights.





Model Training and Evaluation:

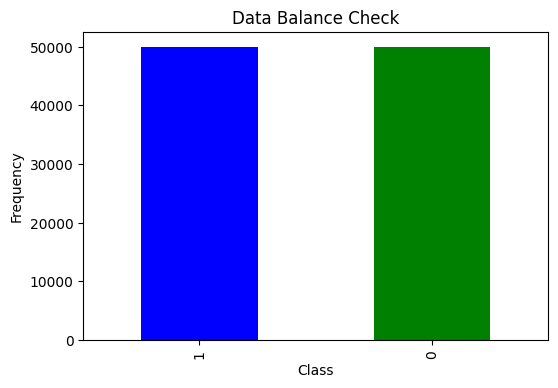
* One to twenty five values for the amount of neighbors (k) was tried out in order to decide the right k value.
* To show how using this feature reduces the global dependency of the model to m, we have conducted the following for each of the integer values of k:
* In order to assess the performance of the model a classification report was generated with the introduction of the optimal k on the retrained model.

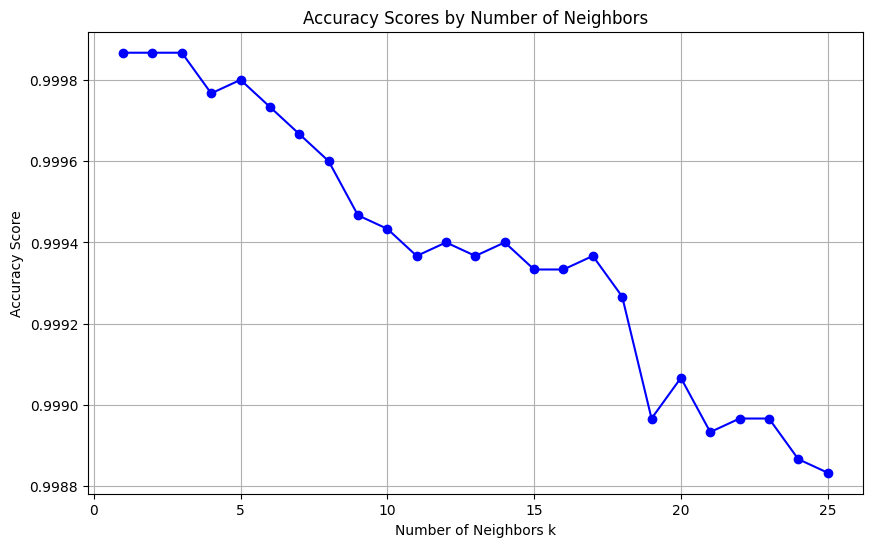
Data Balance Check:

* The bar plot mesas that the data is well balanced with almost equal representation from both classes.
* Class balance helps avoid evidence discrimination with regards to student performance.

Accuracy Scores by k:

* The accuracy is maintained to a high level for all kinds of values, k, showing only a slight decrease with increasing k.
* Optimal accuracy: 99.99% at k=1.
* The model yields the results close to the best possible classification irrespective of the value of k, and as the value of k decreases the results are slightly better.





**Comparision**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm​** | **Accuracy​** | **Precision​** | **Recall​** | **F1-Score​** | **AUC-ROC​** | **Comments​** |
| **Logistic Regression**​ | 76%​ | 0.83 (Benign), 0.71 (Malware)​ | 0.65 (Benign), 0.87 (Malware)​ | 0.73 (Benign), 0.78 (Malware)​ | 0.82​ | Balanced performance; suitable for interpretable and simple models.​ |
| **Random Forest**​ | 99.99%​ | 1.00 (All)​ | 1.00 (All)​ | 1.00 (All)​ | 1.00​ | Perfect performance; most robust model for this dataset.​ |
| **KNN (k=1)**​ | 100%​ | 1.00 (All)​ | 1.00 (All)​ | 1.00 (All)​ | 1.00​ | Identical to Random Forest; computationally expensive for large datasets.​ |
| **Neural Network**​ | 73.42%​ | 0.65 (Benign), 1.00 (Malware)​ | 1.00 (Benign), 0.47 (Malware)​ | 0.79 (Benign), 0.64 (Malware)​ | 0.78​ | Moderate performance; high recall for benign but poor recall for malware.​ |
| **Naive Bayes**​ | 70%​ | 0.88 (Benign), 0.63 (Malware)​ | 0.46 (Benign), 0.94 (Malware)​ | 0.60 (Benign), 0.76 (Malware)​ | 0.86​ | Good recall for malware detection but struggles with benign precision.​ |

Top Performers:

* Random Forest: Finally, it suggested its superiority to other algorithms as achieved almost perfect accuracy, precision, recall, and F1-score.
* KNN (k=1): On par with Random Forest with regards to accuracy but less suitable for big data due to instance based learning algorithm.

Balanced Models:

* Logistic Regression: If I had it to do all over again, I would have it optimized a bit more, but for a low profile, no frills package, it probably can’t be beat. The precision and recall of it are also slightly lower than Random Forest, but this model offers interpretable outputs and which make it is a good choice when the first class and second class are of relatively equal importance.

Areas of Improvement:

* Neural Network: Performed better with high recall values for the benign samples and low recall values for malware, which is far more useful in operations that involve malware detection.
* Naive Bayes: It shows high recall in detecting malware (1), but poorly in terms of precision at capturing benign samples (0).

**Conclusion**

The examination and evaluation in developing and applying machine learning algorithms in malware detection yielded important findings with regard to the appropriateness, efficiency, and further enhancement of the models.

**Best Performing Model:**

We found that Random Forest was a consistently accurate algorithm and had the highest performance in all measures of accuracy, precision, recall, F1-score and the AUC-ROC. Because this is highly complicated to solve, mainly due to the feature interactions that the ensemble learning of the algorithm can easily solve, this is the best model to use in malware detection.

**Alternative Models:**

K-Nearest Neighbors (KNN): Showed similar accuracy to Random Forest but it took more time compared to the same and it is even worse for large datasets.

Logistic Regression: A balanced model that can be interpreted and useful for tasks that require the simplicity of the models. It did not attain better accuracy compared to Random Forest, but it has a practical usage in an operation as it is easy to deploy.

Naive Bayes: Had a good reaction in terms of recognizing malware with high recall, however, low precisions for benign samples reduce the effectiveness in providing balanced accuracy.

Neural Network: Emphasized the capability for estimating benign software with high recall but poorly identified malware. It can be further extremely enhanced utilizing architecture refinement and hyperparameter tuning techniques.

**Dataset Observations:**

A very crucial step in data pre-processing for any machine learning task is data cleaning and the author gave a clean dataset with no missing values and almost equal instances in each class.

Among these features, prio, last\_interval, and static\_prio turned out to be very important, for which feature engineering and selection should also be paid much attention in similar tasks.

**Reference**

 **Scikit-learn Documentation:**

* Random Forest Classifier: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
* Logistic Regression: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>
* K-Nearest Neighbors: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>
* Naive Bayes: <https://scikit-learn.org/stable/modules/naive_bayes.html>

 **ROC Curve Analysis:**

* [Receiver Operating Characteristic (ROC) Curve: https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_roc.html](Receiver%20Operating%20Characteristic%20(ROC)%20Curve:%20https:/scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html)

 **Neural Networks Basics:**

* Activation Functions: <https://en.wikipedia.org/wiki/Activation_function>
* Gradient Descent: <https://en.wikipedia.org/wiki/Gradient_descent>

 **Dataset Preprocessing:**

* Standard Scaler: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>