Introduction to Data Mining

STA 401

Final Project

Malicious Sensor Node Detection

Vibha Bhavikatti - g00089208

Koushal Parupudi - b00087520

Mohammad Arfan Ameen - b00093780

27 November 2023

Fall 2023

American University of Sharjah

Dr. Ayman Alzaatreh

Table of Contents

List of Figures

Abstract

This paper addresses the crucial task of identifying malicious sensor nodes in IoT networks using a dataset comprising 21 features and 10,000 samples. The dataset underwent cleaning, feature selection, and transformation processes to enhance the accuracy of the chosen classification models. The imbalance in the dataset was mitigated through a combination of over- and under-sampling techniques(SMOTE). Feature selection involved consulting scatterplots, correlation matrices, and Principal Component Analysis (PCA) to identify key predictors. Four models—Random Forest, Support Vector Machines (SVM), XGBoost, and Long Short-Term Memory (LSTM)—were employed and evaluated based on accuracy, precision, recall, and F1-score. The results showcase the superior performance of XGBoost and Stacked LSTM, achieving accuracy rates of 99.83% and high precision, recall, and F1-scores. SVM, with Radial and Linear kernels, demonstrated commendable performance, while Random Forest slightly lagged behind. Recommendations include considering model selection based on computational resources and application requirements, validating models on real-world data, and exploring ensemble methods for improved performance. Overall, this study emphasizes the efficacy of advanced machine learning models in enhancing IoT network security by accurately identifying malicious nodes.

Keywords: IoT Networks; Malicious Sensor Nodes Detection; Random Forest; SVM; XGBoost; LSTM.

1.Introduction

IOT (Internet Of Things) is a fast growing industry that has played an important role in the deployment of sensor networks in smart homes and industrial automation. However, this has introduced the challenge of identifying nodes within these networks that have been compromised by malicious attackers. These attacks can range from blackhole attacks, gray hole attacks, flooding attacks, sybil attacks, each requiring a unique response to neutralize the threat. The diversity of the metrics in this dataset allows for clear understanding of node behavior within these networks under different circumstances. Being able to identify accurately the sensor nodes that were subject to attackers, is critical as it allows action to be taken quicker to maintain network integrity.

In the context of securing these networks, accurately identifying sensor nodes that have been compromised by attackers holds critical importance. Such precise identification enables swift actions to maintain the integrity of the entire network. To address this critical binary classification of malicious sensor nodes problem, we employ a diverse set of advanced machine learning models, including Random Forest, Support Vector Machines (SVM), XGBoost, and Long Short-Term Memory (LSTM) networks.

Throughout the report we will be evaluating the performance of the aforementioned models using appropriate evaluation metrics such as: Confusion matrix, from which we can extract the Accuracy, Precision, Recall and F1 scores. Additionally, we plotted the ROC (Receiver Operating Characteristic curve) and obtained the AUC (Area under ROC curve). The results of this project can provide insight for network security and optimize response time for maintaining the integrity of the network.

1.1.Literature Review

The detection of malicious nodes in wireless sensor networks is an essential aspect to ensure the security and reliability of data transmission. Two research papers propose the K-clustering and J-48 tree algorithm for the detection of malicious sensor nodes. In this literature review, we explored the K-clustering and J-48 tree algorithms to see their effectiveness in this classification problem.

In the first paper, the methodology proposes a 2 stage malicious sensor node detection system [1]. A trace file is generated using Network simulator 2 and is used as a dataset. The dataset consists of 120 nodes, on which in the initial stage the K-means clustering algorithm is applied on each data point and the centroid of the cluster the points belong to. The algorithm iteratively assigns the data point to the cluster with the closest mean and then updates the mean of the cluster. In the second stage, the J48 decision tree algorithm is applied for solving the classification problem. A decision tree is built based on the training dataset and the algorithm operates recursively on the attributes of the nodes to be able to classify into low, medium or high.

In the second paper, the proposed mechanism for detecting black hole attacks in wireless sensor networks also uses the K-clustering and J-48 algorithm for classification with a simulator called NS2 [2]. They also conducted the network's analysis using an Awk file. They use two scenarios, one with 100 nodes and another with 115 nodes. The first network is without the malicious activities and the other network included nodes that depicted malicious behavior. The K-Means clustering algorithm is an efficient clustering algorithm where clusters of data items are made based on their distance cluster heads or means. A center for each of the clusters is created such that the data items within the cluster are mutually farthest apart from each other. Each data item's minimum distance is checked and is assigned to the centers. Lastly, a mean or center position is recalculated every time a new data item is added to the cluster and this process is repeated until the final set of clusters are formed. Since the aim of the paper was to detect black hole attacks, it was found that nodes with maximum receiving ratio and zero forwarding ratio were black hole or malicious nodes as the packets were not reaching the destination and were dropped instead.

Moreover, the paper by Muniyandi et al. discusses methods to identify attacks or malicious activity in a network [3]. The methods used are K-means clustering and decision trees. The dataset they used was the KDD99 dataset which is a generally used dataset for testing anomaly detection models. The K-means clustering is used to partition the training instances into k clusters using the Euclidean distance similarity. Following this, the authors build decision trees using the C4.5 decision tree algorithm on each cluster , which represents a density region of normal or anomaly instances. This refines the decision boundary of each cluster using the subgroups within the cluster. They were able to conclude that using K-means clustering and C4.5 Decision Trees gives the highest precision when compared to K-means, ID3, Naive Bayes, K-NN, SVM and TCM-KNN.

Lastly, the paper “Wireless Sensor Networks Intrusion Detection Based on SMOTE and the Random Forest Algorithm” [4] proposes to balance the commonly used KDD99 using the SMOTE (Synthetic Minority Oversampling Technique) and then uses a random forest algorithm to train the classifier for intrusion detection. SMOTE works by creating “artificial” data points along the line segments connecting the minority class instance to its KNN. These synthetic instances are placed into the original dataset to remove the imbalance in data. The authors found that using the random forest algorithm alone they achieve a 92.39% accuracy rate, which was found to be higher than the accuracy rates of other methods like J48, LibSVM, NaiveBayes,Bagging and AdaboostM1. This is further increased when SMOTE is applied to the dataset, increasing the accuracy rate to 92.57%.

2.Statistical Methods Used

Random Forest:

Decision Trees: It is a supervised learning method used for both regression and classification tasks. The algorithm creates subsets of the dataset by splitting it based on the important features, forming a tree-like structure where each internal node represents a decision based on an important feature. Each branch represents an outcome of the split and the leaf nodes are the results of the decision tree, i.e the final prediction. The internal nodes are split to maximize information gain for classification trees. This process is repeated until a stopping criterion is met - reaching a specified depth or until there remains too few samples.

Random Forest: It is an ensemble learning method that uses multiple decision trees using different subsets of the training data and makes predictions by calculating the mode of the predictions of individual trees for classification and averaging for regression. It additionally provides information about the variable importance that could aid in feature selection.

Support Vector Machines (SVM):

SVMs are a set of supervised learning methods adept in solving classification and regression problems. They are especially effective on high-dimensional datasets. SVMs also display high memory efficiency because of their ability to focus on a subset of training points in the decision function.

One of the advantages of SVMs lies in their versatility, which stems from the availability of various kernel functions. These kernel functions allow SVMs to adapt to the specific characteristics of the given datasets. The kernel functions include Linear, Polynomial (Degree of 3), Radial and Sigmoid kernels which are tailored to the different possible trends in a dataset, enhancing the model's flexibility and performance across diverse scenarios.

XGBoost:

eXtreme Gradient Boosting (XGBoost) is a more efficient implementation of gradient boosting algorithms. Based on the method of gradient boosting, new models are added in a sequential manner to address the errors made by existing models. XGBoost is particularly known for its speed and performance, especially in structured or tabular data. This has gained popularity in recent years as it enhances the performance by introducing several changes, such as regularization techniques, parallel processing, cross-validation at each iteration and an efficient tree pruning approach.

LSTM:

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) architecture used in deep learning. In this architecture, the problem of vanishing gradients in RNN is addressed and an added benefit is that the model holds the history of the previous input and uses it for the processing of the current input. LSTMs are especially useful for large datasets and work well for sequential data. Hence, it is beneficial to use when there is a need for learning long term dependencies between the time steps in the data. An extension of the LSTM model is the Stacked LSTMs, where it basically consists of multiple LSTM layers, where each layer consists of multiple memory cells. This helps in increasing the depth of the model as well.

3.Data Description

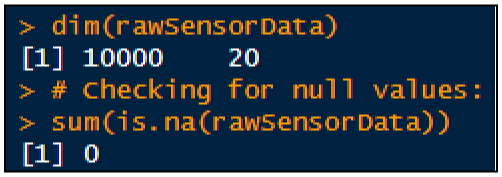
The dataset for identifying malicious sensor nodes is taken from the IEEE DataPort website. This dataset contains 21 features and 10000 samples, which can be used for prediction of malicious nodes. The goal is to create models that classify unseen sensor data as malicious or not, i.e. sensor nodes compromised due to malicious attacks. The variables in this dataset are as illustrated in Table 1.

Variables	**Description**	**Units**
Node ID	The unique identifier for each node	--
Timestamp	The time at which a data packet is sent or received	--
IP Address	The Internet Protocol address of a node	--
Packet Rate	Rate of transmission of data packets	Kbps
Packet Drop Rate	Rate at which packets are being dropped	--
Packet Duplication Rate	Rate at which the packets are duplicated	--
Data Throughput	The amount of data transmitted from one point to another in a time frame	Kbps
Signal Strength	Power level of the signal	dBm
Signal-to-Noise Ratio (SNR)	Clarity of the signal	dB
Battery Level	Remaining power level of the node	--
Energy Consumption Rate	Rate of consumption of energy by the nodes	KWh
Number of Neighbors	Number of nodes within direct communication range	--

Table 1: Description of the 21 variables from the dataset

4.Cleaning Methods

The cleaning involves finding and deleting any duplicate data, as this would lead to more accurate and consistent data, and thus a more accurate prediction. Incorrect data would negatively affect the performance of the model. We also aim to remove unwanted features that would not be helpful in the analysis. Minor changes to the dataset were made as shown below:



```
> dim(rawSensorData)
[1] 10000    20
> # Checking for null values:
> sum(is.na(rawSensorData))
[1] 0
```

Figure 1: Dimension and sum of null values

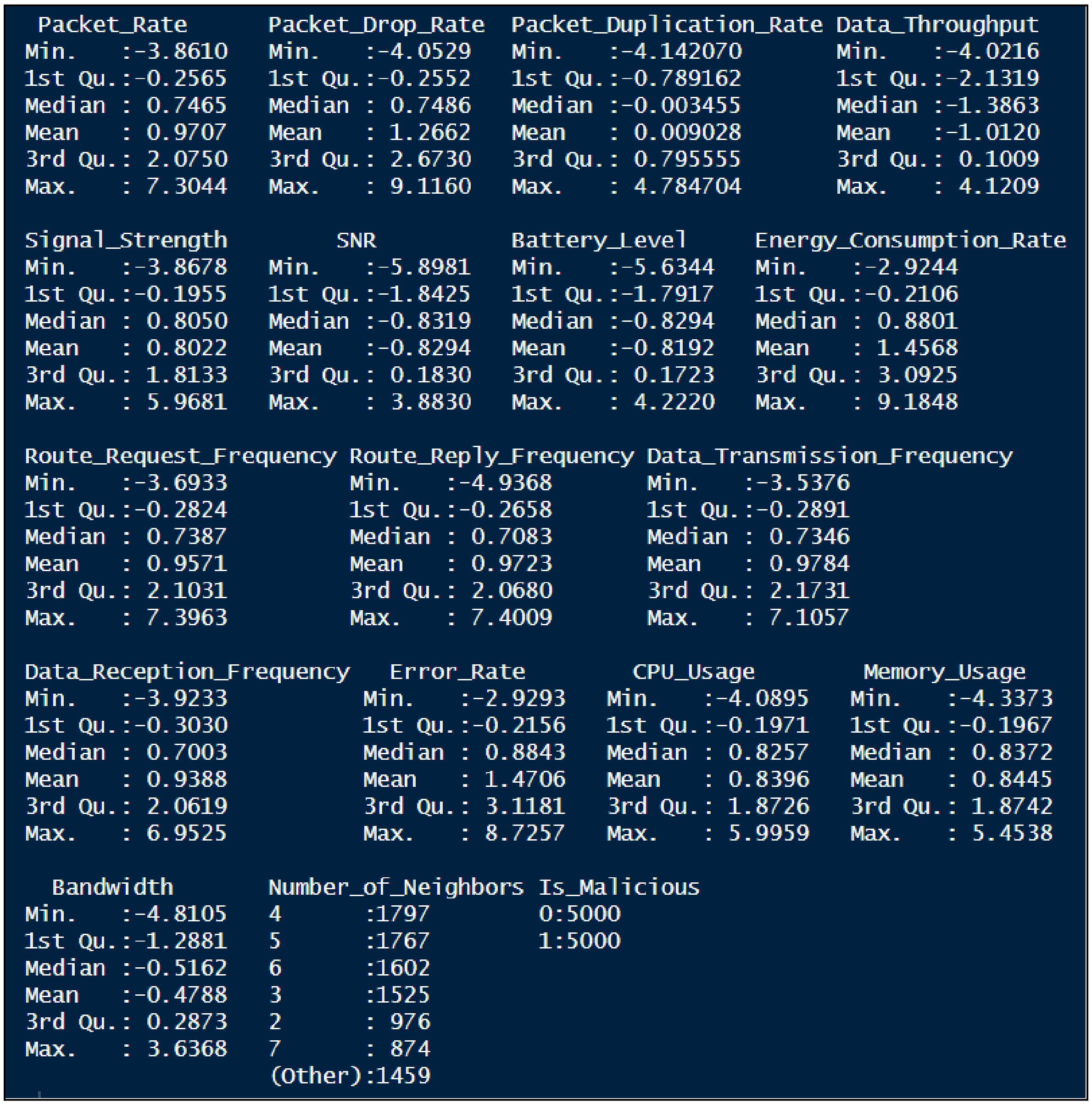
The dimensions of the dataset are 10000 rows and 20 columns. Using the 'is.na' command, we can see that there are no NA data or duplicates in the dataset. Through the 'distinct' command which keeps the distinct rows in the data, we confirmed that there were no duplicates. We removed two variables: IP Address and Timestamp as it did not serve any purpose in the process of classification. This will ensure that the dataset contains no duplicates that can lead to inaccurate analyses and modeling. For instance, if a record appears multiple times, it may contribute more weight to certain patterns or trends than it should.

5. Data Transformation

Data transformation for the SensorNetGuard dataset is initially involved in encoding the timestamp variable into 4 classes: "Morning", "Afternoon", "Evening" and "Night". However we ascertained that this variable does not hold importance in the classification of malicious nodes and hence we decided to remove it from the dataset. We also removed the Node ID and IP Address columns as the former was redundant and the latter did not serve any purpose for the classification problem.

Furthermore, we standardized the dataset to ensure that all variables are on a similar scale. Since the data has variables of different units, it is essential to perform this preprocessing step. This will avoid a certain feature dominating smaller features in the analysis and machine learning process as the model may tend to focus on optimizing the dominant feature and neglect the smaller features. The data was standardized using the scale() function, which does z-score standardization. Here, the data is standardized to have a mean of zero and a standard deviation of 1. In the context of the methods we are using for feature selection, standardizing the data will allow the PCA to represent the contribution of all features accurately.

Additionally, addressing imbalance in the dataset is an essential data preprocessing step as it heavily affects the results when different models are tested. This is performed to prevent model bias towards the majority class, improve generalization for the minority class, and ensure accurate performance evaluation. When evaluating standard metrics like accuracy, this can be misleading if the data is imbalanced as it then leads to poor model performance, especially on the minority class. Hence, creating more samples of the minority class and reducing samples of the majority class helps in mitigating the imbalance in the dataset. This is especially crucial in real-world applications where the minority class is of significant importance, such as in fraud detection, where misclassifying rare cases can have severe consequences.



```
  Packet_Rate       Packet_Drop_Rate  Packet_Duplication_Rate Data_Throughput
 Min.   :-3.8610   Min.   :-4.0529   Min.   :-4.142070        Min.   :-4.0216
 1st Qu.:-0.2565   1st Qu.:-0.2552   1st Qu.:-0.789162        1st Qu.:-2.1319
 Median : 0.7465   Median : 0.7486   Median :-0.003455        Median :-1.3863
 Mean   : 0.9707   Mean   : 1.2662   Mean   : 0.009028        Mean   :-1.0120
 3rd Qu.: 2.0750   3rd Qu.: 2.6730   3rd Qu.: 0.795555        3rd Qu.: 0.1009
 Max.   : 7.3044   Max.   : 9.1160   Max.   : 4.784704        Max.   : 4.1209

 Signal_Strength        SNR          Battery_Level     Energy_Consumption_Rate
 Min.   :-3.8678   Min.   :-5.8981   Min.   :-5.6344   Min.   :-2.9244
 1st Qu.:-0.1955   1st Qu.:-1.8425   1st Qu.:-1.7917   1st Qu.:-0.2106
 Median : 0.8050   Median :-0.8319   Median :-0.8294   Median : 0.8801
 Mean   : 0.8022   Mean   :-0.8294   Mean   :-0.8192   Mean   : 1.4568
 3rd Qu.: 1.8133   3rd Qu.: 0.1830   3rd Qu.: 0.1723   3rd Qu.: 3.0925
 Max.   : 5.9681   Max.   : 3.8830   Max.   : 4.2220   Max.   : 9.1848

 Route_Request_Frequency Route_Reply_Frequency Data_Transmission_Frequency
 Min.   :-3.6933         Min.   :-4.9368       Min.   :-3.5376
 1st Qu.:-0.2824         1st Qu.:-0.2658       1st Qu.:-0.2891
 Median : 0.7387         Median : 0.7083       Median : 0.7346
 Mean   : 0.9571         Mean   : 0.9723       Mean   : 0.9784
 3rd Qu.: 2.1031         3rd Qu.: 2.0680       3rd Qu.: 2.1731
 Max.   : 7.3963         Max.   : 7.4009       Max.   : 7.1057

 Data_Reception_Frequency   Error_Rate         CPU_Usage         Memory_Usage
 Min.   :-3.9233          Min.   :-2.9293   Min.   :-4.0895   Min.   :-4.3373
 1st Qu.:-0.3030          1st Qu.:-0.2156   1st Qu.:-0.1971   1st Qu.:-0.1967
 Median : 0.7003          Median : 0.8843   Median : 0.8257   Median : 0.8372
 Mean   : 0.9388          Mean   : 1.4706   Mean   : 0.8396   Mean   : 0.8445
 3rd Qu.: 2.0619          3rd Qu.: 3.1181   3rd Qu.: 1.8726   3rd Qu.: 1.8742
 Max.   : 6.9525          Max.   : 8.7257   Max.   : 5.9959   Max.   : 5.4538

   Bandwidth         Number_of_Neighbors Is_Malicious
 Min.   :-4.8105   4      :1797          0:5000
 1st Qu.:-1.2881   5      :1767          1:5000
 Median :-0.5162   6      :1602
 Mean   :-0.4788   3      :1525
 3rd Qu.: 0.2873   2      : 976
 Max.   : 3.6368   7      : 874
                   (Other):1459
```

Figure 2: Summary of the Dataset

In our project, the ovun.sample function from the ROSE library was used to primarily balance our dataset. The main issue in the dataset is the imbalance of the **y** variable, Is_Malicious, as there were 9513 rows with ‘0’ as the value for Is_Malicious and 487 rows with ‘1’ respectively. This affected the results of the methods we planned to use and therefore, resulted in being biased towards the class ‘0’. Hence, this is why we performed a combination of over- and under-sampling using the ovun.sample function, where the issue of the imbalance in the dataset was rectified.

6.Feature Selection

As the sensor nodes dataset consists of 21 variables, it is important to select features that will help in performing more efficient classification methods. Fig 1 below depicts the scatterplot of the quantitative variables in our dataset. However, due to the high number of values in the dataset, it is slightly difficult to determine the correlation between the variables.

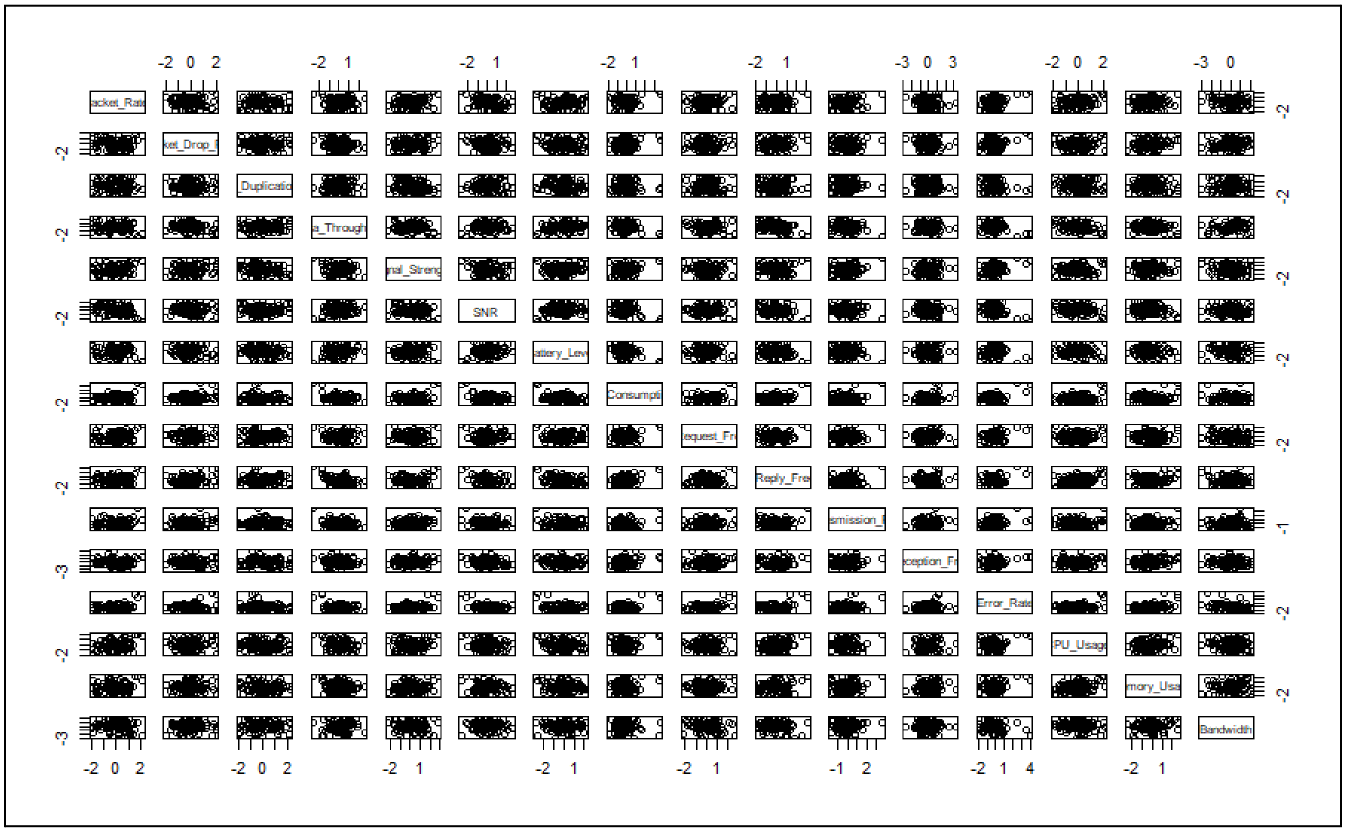


Figure 3: Scatterplot of the dataset

Therefore we used a Correlation matrix of the quantitative variables to accurately represent the correlation between the variables as seen in Fig 2 below.

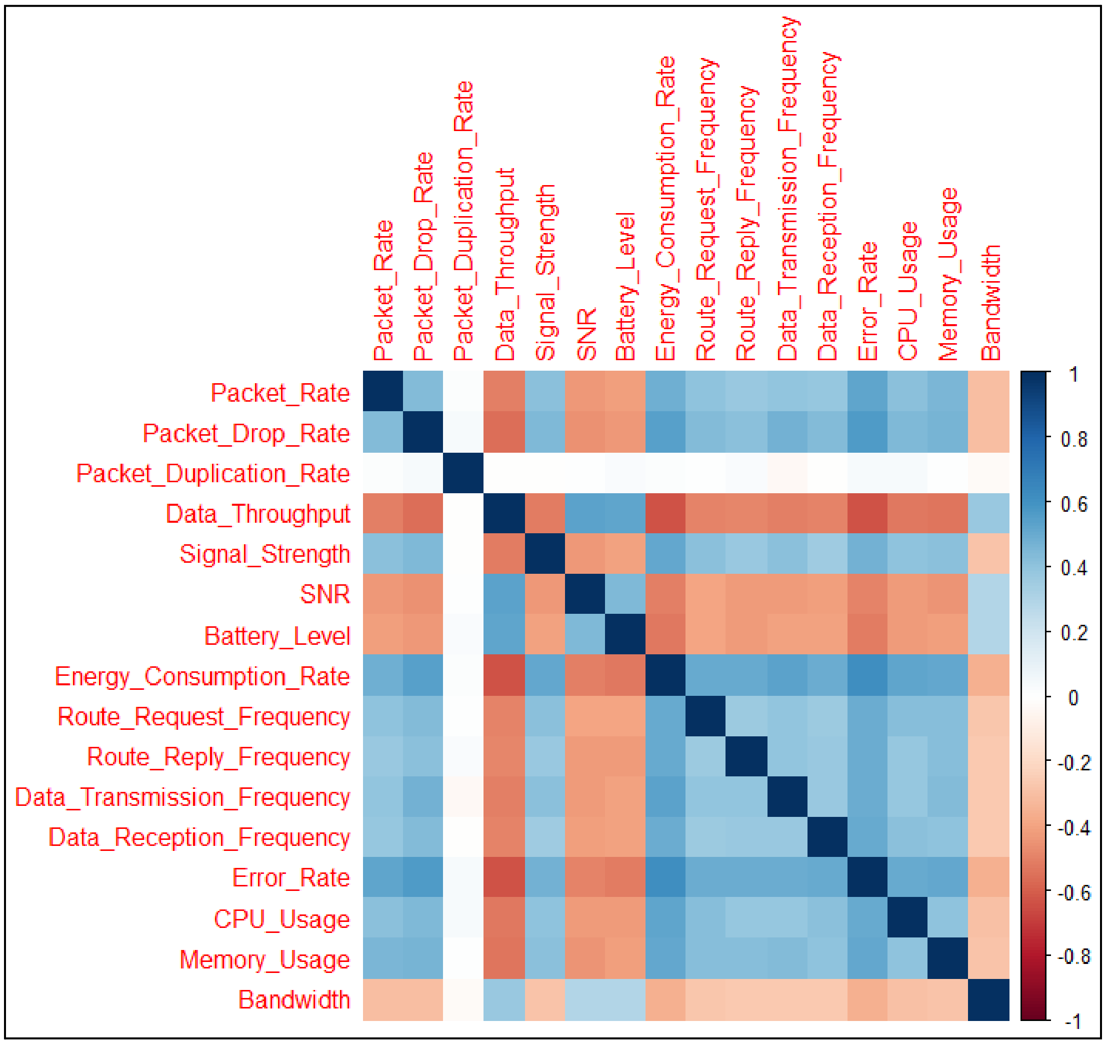
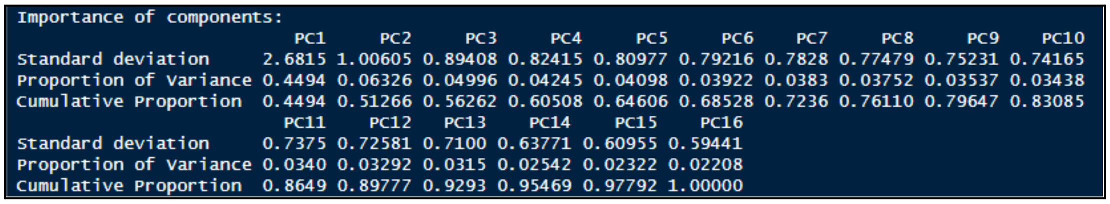


Figure 4: Correlation matrix

Based on the above figure, we can conclude that the Packet duplication rate variable exhibits no linear correlation with any of the other variables. This however, does not mean that the variable can be excluded from the dataset as it may have a nonlinear relationship with the other predictors.

Principal Component Analysis (PCA) was used to help us reduce the dimensionality of our data set as well as for feature selection. PCA achieves this by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Principal Components are new variables that are constructed as linear combinations or mixtures of the initial variables. This is done in such a way that the PCs are uncorrelated and most of the info within the initial PCs are squeezed and compressed into the first components.
After finding the principal components we analyzed the importance of the PCs (Figure 3),,the Scree plot (Figure 4) which explains how much of the variance in the dataset is explained by that PC and the Plot of the variables in respect to the variability they represent (Figure 5).



```
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC6    PC7     PC8     PC9    PC10
Standard deviation     2.6815 1.00605 0.89408 0.82415 0.80977 0.79216 0.7828 0.77479 0.75231 0.74165
Proportion of Variance 0.4494 0.06326 0.04996 0.04245 0.04098 0.03922 0.0383 0.03752 0.03537 0.03438
Cumulative Proportion  0.4494 0.51266 0.56262 0.60508 0.64606 0.68528 0.7236 0.76110 0.79647 0.83085
                         PC11    PC12   PC13    PC14    PC15    PC16
Standard deviation     0.7375 0.72581 0.7100 0.63771 0.60955 0.59441
Proportion of Variance 0.0340 0.03292 0.0315 0.02542 0.02322 0.02208
Cumulative Proportion  0.8649 0.89777 0.9293 0.95469 0.97792 1.00000
```

Figure 5: Importance of Principal Components

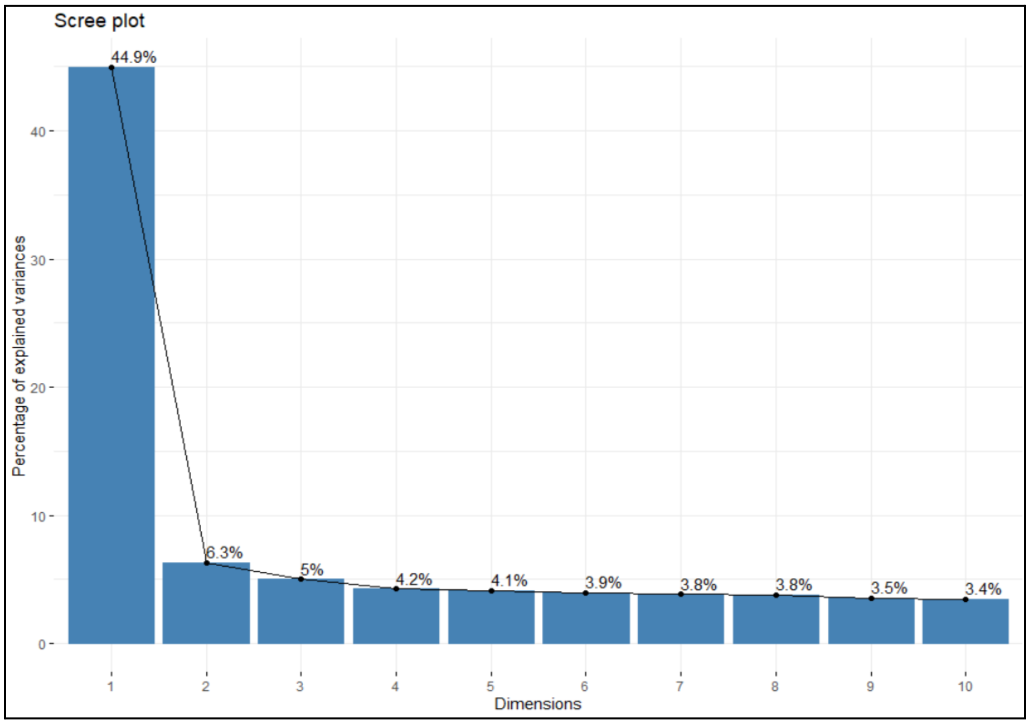


Figure 6: Scree Plot of PCAs

As we can see from Figure 4, PC1 (First Principal Component) explains 44.9% of the variance in the dataset, with the next PC only explaining 6.3% of variance. This forms an “elbow” in the scree plot which, using Kaiser Criterion, suggests that only using PC1 will allow us to capture the majority of the variance in the dataset.

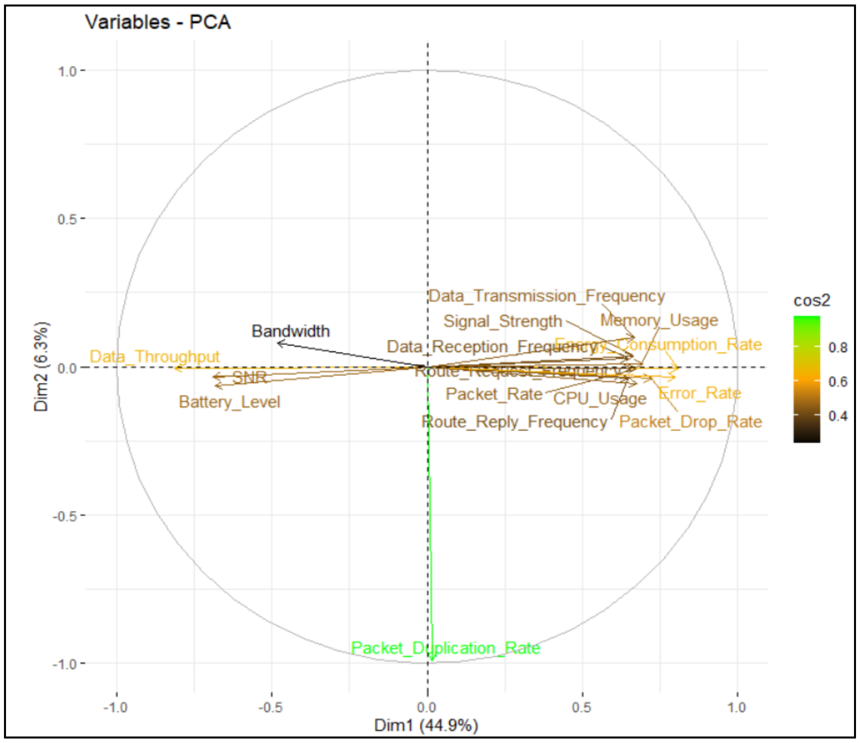


Figure 7: PCA plot of variables

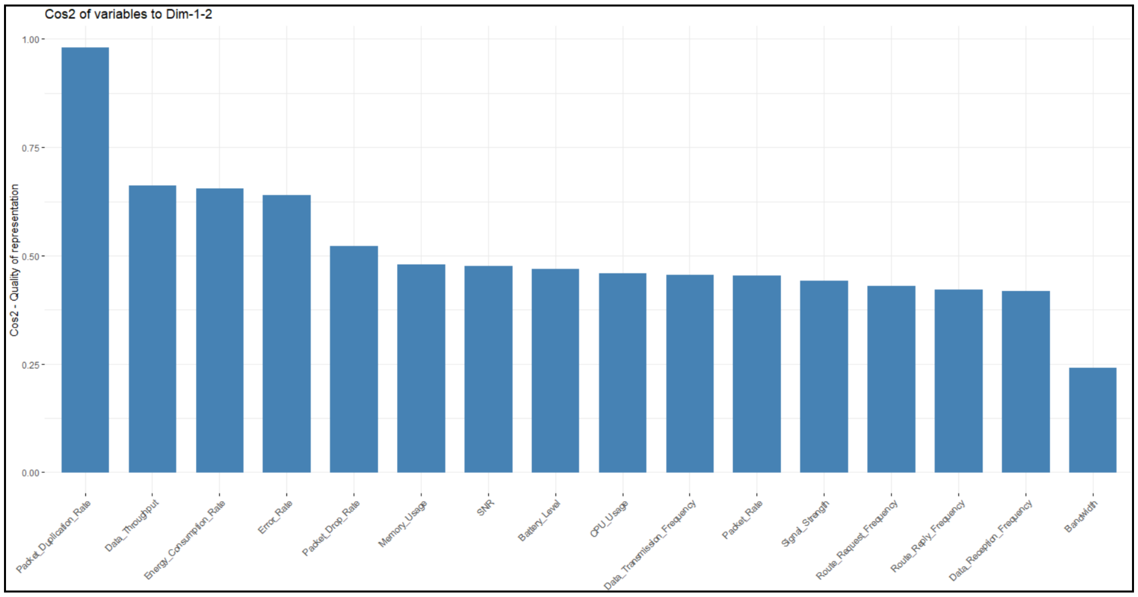


Figure 8: COS2 Vs Variable Names Plot

From Figure 5, we can see that Packet_Duplication_Rate, Data_Throughput, Energy_Consumption_Rate, and Error_Rate have the highest COS2 in that order. COS2 is the square of the cosine and it refers to the quality of representation of that variable by the PCA. A higher value implies a better representation while a lower value indicates that it is not properly represented by the PCA.

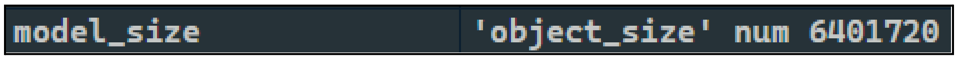
Thus, we can narrow down our choice of variables to Packet_Duplication_Rate, Data_Throughput, Consumption_Rate, and Error_Rate. However, upon further testing with our statistical models, we settled on using only 2 predictors, i.e. Packet_Duplication_Rate and Data_Throughput, in order to predict whether or not a node is malicious or not.

This selection is further backed by the fact that Packet_Duplication_Rate represents how many packets are duplicated which is caused by the failure in transmitting the packets (Packet Loss). When a node is malicious it is logical to assume that there will be a correlation with Packet_Duplication_Rate. In addition to that we can also see a connection between Data_Throughput and malicious nodes as malicious nodes will affect the network system and thus affect the Data_Throughput of each node.

7.Model Approach and Results

Random Forest:

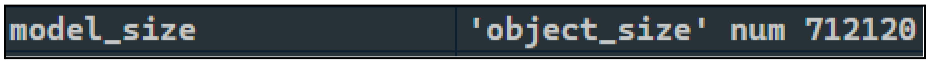
When creating the random forest, we ensured to split the dataset into 60% training and 40% testing (of 10000 data points). Initially when performing the random forest model testing, we noticed that it was very computationally expensive due to the large number of datapoints. This is evident from the model size as shown below.



```
model_size                    'object_size' num 6401720
```

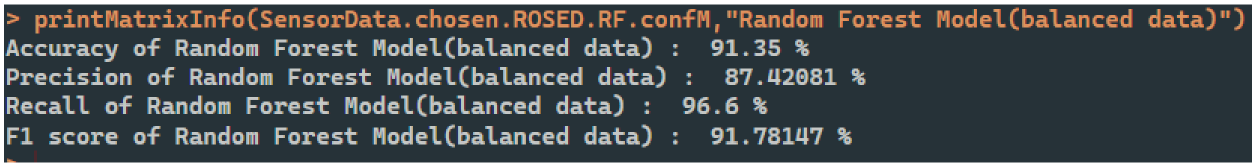
Figure 9: Size of Random Forest Model

Therefore, in the interest of reducing the computational expense, we decided to train the random forest model using a small subset of the training dataset. This created a model that was nearly 90% smaller than the original model without greatly affecting the accuracy.



```
model_size                    'object_size' num 712120
```

Figure 10: Size of Random Forest Model (Subset)



```
> printMatrixInfo(SensorData.chosen.ROSED.RF.confM,"Random Forest Model(balanced data)")
Accuracy of Random Forest Model(balanced data) :  91.35 %
Precision of Random Forest Model(balanced data) :  87.42081 %
Recall of Random Forest Model(balanced data) :  96.6 %
F1 score of Random Forest Model(balanced data) :  91.78147 %
```

Figure 11: Evaluation of the Random Forest Model

The figure above shows the results of running the random forest for classification of malicious sensor nodes after training it using a smaller subset of the original training dataset. The accuracy indicates that the model correctly classified malicious and non-malicious nodes nearly 91.35% of the time. The precision indicates that when the model classifies that a node is malicious, it is correctly doing so 87.42% of the time. High precision is important as there is more importance in classifying a malicious node to maintain network integrity. Having a high recall and high precision go hand in hand in identifying malicious nodes. Most importantly however, is the F1 score, which indicates a balance in the precision and recall, as well as that the model identifies the malicious nodes and minimizes the false positives 91.8% of the time. The results indicate that the Random Forest model trained on a balanced dataset is robust in classifying malicious nodes. The high accuracy, precision, recall, and F1 score demonstrate the effectiveness of the model in maintaining a balance between correctly classifying malicious nodes and avoiding false positives.

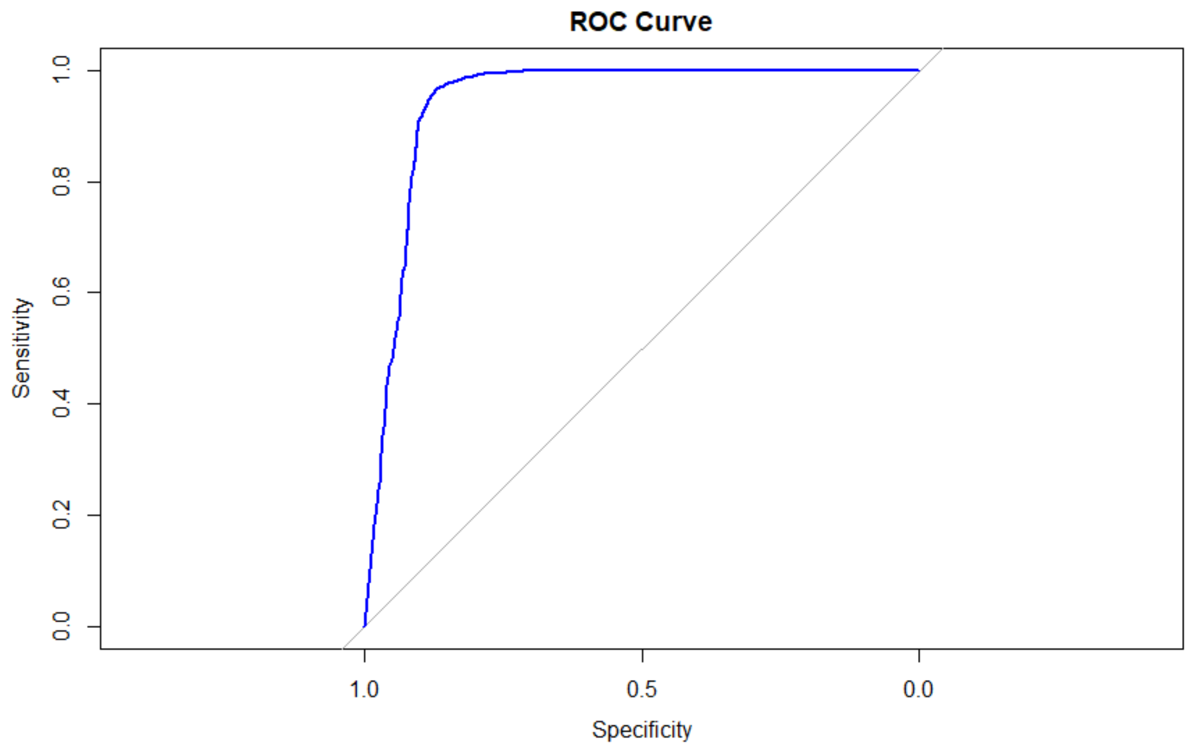
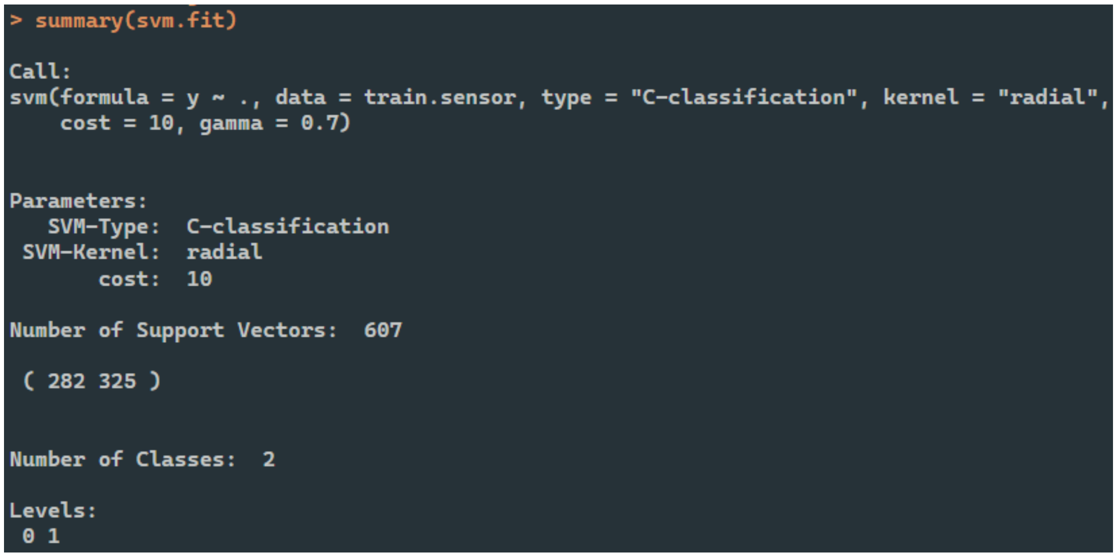


Figure 12: ROC Curve for Random Forest

The ROC curve indicates that the Random forest model classifies the malicious nodes pretty well. The AUC score being 0.94 also is an indication that the model discriminates well between the malicious and non-malicious nodes.

SVM:

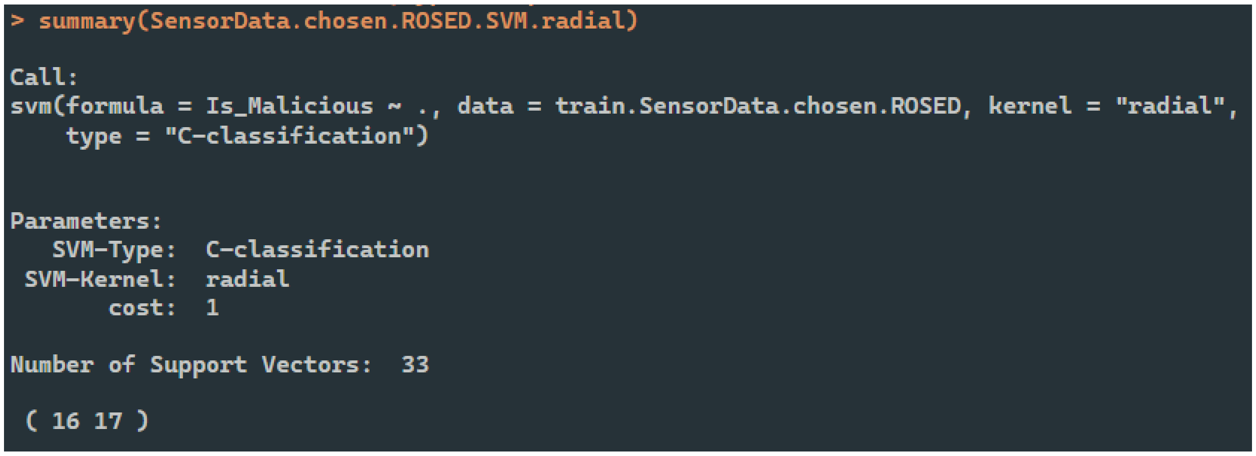
When creating the SVM model, we ensured to split the dataset into 60% training and 40% testing (of 10000 data points). Initially when performing the SVM model fitting, we noticed that it was very computationally expensive due to the large number of support vectors.



```
> summary(svm.fit)

Call:
svm(formula = y ~ ., data = train.sensor, type = "C-classification", kernel = "radial",
    cost = 10, gamma = 0.7)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  607

 ( 282 325 )


Number of Classes:  2

Levels:
 0 1
```

Figure 13: SVM Radial Model Summary

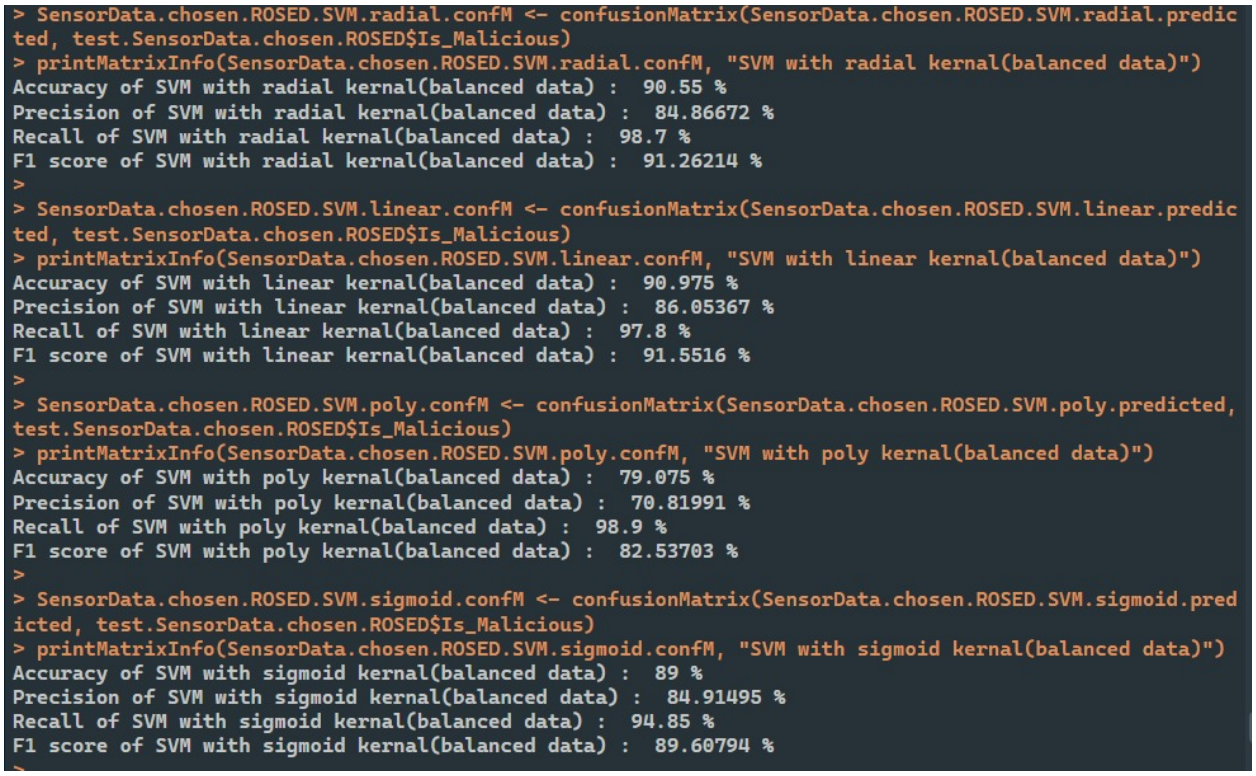
From the above fig we can see that the model creates 607 support vectors, which indicates that there is much computational expense from fitting the model. As done previously, we decided to use a subset of the training data which helped alleviate the stress on the CPU and thereby reduce the computational cost.



```
> summary(SensorData.chosen.ROSED.SVM.radial)

Call:
svm(formula = Is_Malicious ~ ., data = train.SensorData.chosen.ROSED, kernel = "radial",
    type = "C-classification")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  33

 ( 16 17 )
```

Figure 14: SVM Radial Model Summary (Subset)

Finally, we fitted the model in the 4 kernel settings: Linear, Radial, Poly and Sigmoid, to get a comparative analysis of the performance of the SVM model.



```
> SensorData.chosen.ROSED.SVM.radial.confM <- confusionMatrix(SensorData.chosen.ROSED.SVM.radial.predic
ted, test.SensorData.chosen.ROSED$Is_Malicious)
> printMatrixInfo(SensorData.chosen.ROSED.SVM.radial.confM, "SVM with radial kernal(balanced data)")
Accuracy of SVM with radial kernal(balanced data) :  90.55 %
Precision of SVM with radial kernal(balanced data) :  84.86672 %
Recall of SVM with radial kernal(balanced data) :  98.7 %
F1 score of SVM with radial kernal(balanced data) :  91.26214 %
>
> SensorData.chosen.ROSED.SVM.linear.confM <- confusionMatrix(SensorData.chosen.ROSED.SVM.linear.predic
ted, test.SensorData.chosen.ROSED$Is_Malicious)
> printMatrixInfo(SensorData.chosen.ROSED.SVM.linear.confM, "SVM with linear kernal(balanced data)")
Accuracy of SVM with linear kernal(balanced data) :  90.975 %
Precision of SVM with linear kernal(balanced data) :  86.05367 %
Recall of SVM with linear kernal(balanced data) :  97.8 %
F1 score of SVM with linear kernal(balanced data) :  91.5516 %
>
> SensorData.chosen.ROSED.SVM.poly.confM <- confusionMatrix(SensorData.chosen.ROSED.SVM.poly.predicted,
test.SensorData.chosen.ROSED$Is_Malicious)
> printMatrixInfo(SensorData.chosen.ROSED.SVM.poly.confM, "SVM with poly kernal(balanced data)")
Accuracy of SVM with poly kernal(balanced data) :  79.075 %
Precision of SVM with poly kernal(balanced data) :  70.81991 %
Recall of SVM with poly kernal(balanced data) :  98.9 %
F1 score of SVM with poly kernal(balanced data) :  82.53703 %
>
> SensorData.chosen.ROSED.SVM.sigmoid.confM <- confusionMatrix(SensorData.chosen.ROSED.SVM.sigmoid.pred
icted, test.SensorData.chosen.ROSED$Is_Malicious)
> printMatrixInfo(SensorData.chosen.ROSED.SVM.sigmoid.confM, "SVM with sigmoid kernal(balanced data)")
Accuracy of SVM with sigmoid kernal(balanced data) :  89 %
Precision of SVM with sigmoid kernal(balanced data) :  84.91495 %
Recall of SVM with sigmoid kernal(balanced data) :  94.85 %
F1 score of SVM with sigmoid kernal(balanced data) :  89.60794 %
>
```

Figure 15: SVM Model Evaluations

From figure above, we can determine that the best overall performance was observed in the Radial and Linear Kernels SVM. These models outperform the others, achieving the highest overall performance. The models effectively identify a larger proportion of actual malicious nodes while maintaining good precision and follow closely with a balanced performance across accuracy, precision, recall, and F1 score. Polynomial and Sigmoid Kernel SVMs show lower accuracy and precision, suggesting they might struggle with capturing a sufficient number of actual malicious nodes.

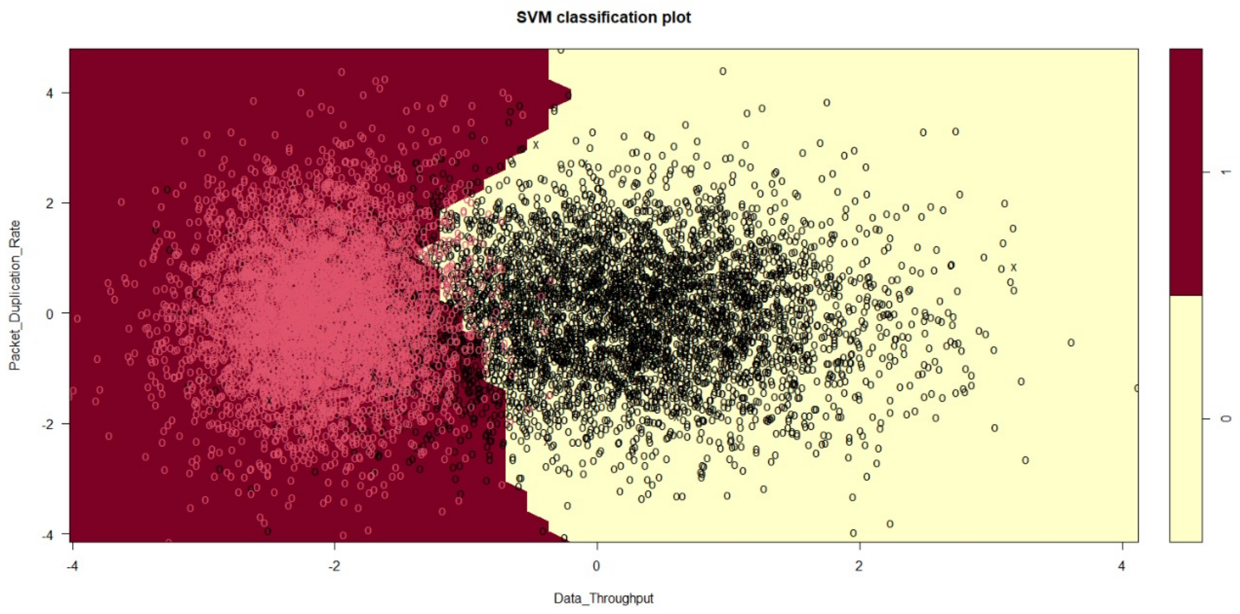


Figure 16: SVM Radial Kernel Model

Figure 15 depicts a plot of the kernel type 'radial' from the SVM model implemented. This plot shows a two-dimensional feature space where the two classes of data points are marked by 'x' and 'o'. The horizontal axis is the 'Data_Throughput' and the vertical axis is the 'Packet_Duplication_Rate' variable. A dense cluster of data points is centered around the origin, and there is an apparent overlap of classes near the boundary, which can mean that there is some misclassification or non-linearity in the relationship between features and classes. The plot has two distinct colored areas, each representing the corresponding class. The red area indicates one class, and the yellow area indicates the other class. From the plot, it can be deduced that the support vectors would be the points closest to the decision boundary. These are the data points that the SVM uses to maximize the margin between classes. Overall, the model appears to differentiate between classes, but a quantitative analysis is needed to assess its generalization and performance metrics.

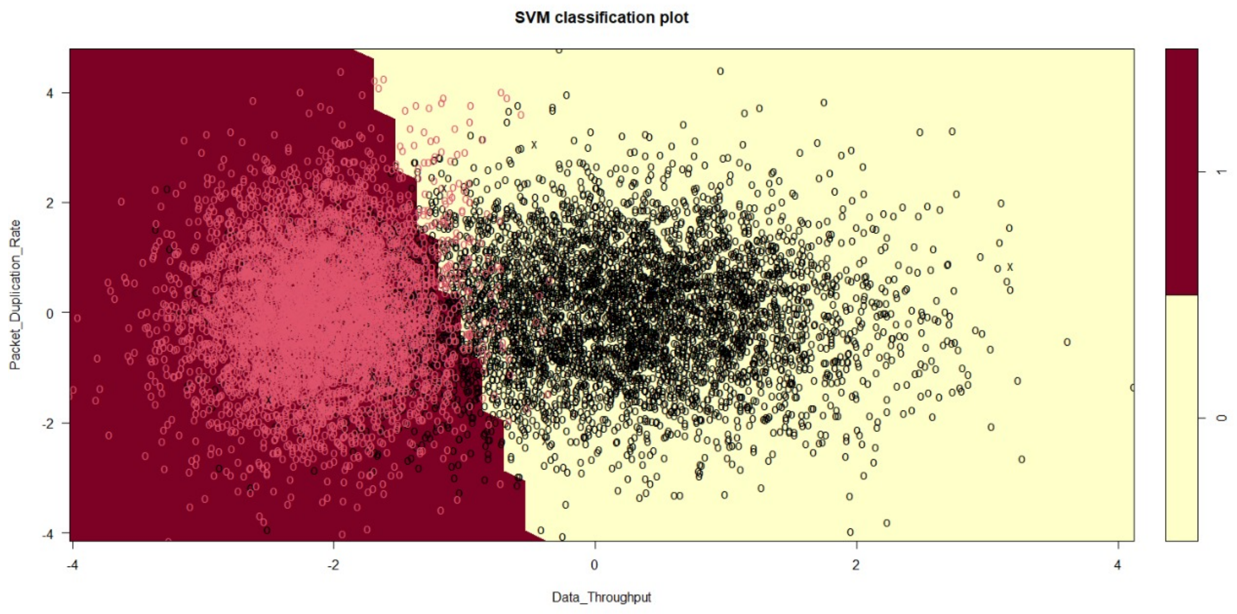


Figure 17: SVM Linear Kernel Model

Figure 16 is the plot of SVM kernel type 'linear'. It is similar to the radial kernel plot except here there is a decision boundary between two classes in a feature space defined by "Data_Throughput" and "Packet_Duplication_Rate". The red and yellow areas denote the SVM's classification regions, where the two classes' data points overlap, indicating potential misclassifications. The plot suggests that while the SVM has learned to separate the classes as well, with red indicating one class and yellow the other.

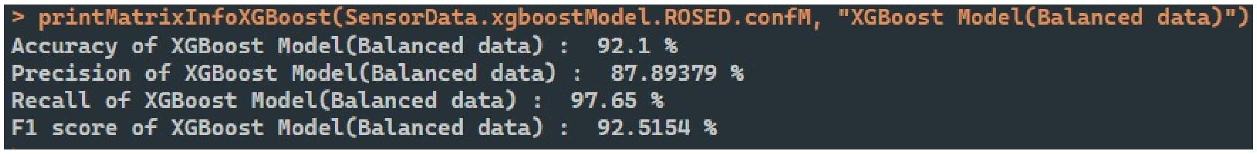
XGBoost:

We used the library "xgboost" in order to create our model. This library comes with a cross-validation function to select the most optimal and efficient model.
In order to do cross validation and train a xgboost model we have to define a set of parameters. We decided on the following parameters:

Parameter Name:	**Parameter Value:**	**Explanation & Reason:**
Booster	gbtree	Specifies the type of boosting model to be used.

		“Gbtree” indicates that decision trees must be used as base learners.
objective	binary:logistic	Defines the learning task and the corresponding learning function. “Binary:logistic” specifies binary classification with logistic regression as the objective.
eta	0.3	Represents the learning rate and controls the step size shrinkage during each boosting iteration. A lower rate usually requires more iterations but can lead to better model generalization
max_depth	6	Specifies the maximum depth of each decision tree by limiting the number of nodes in each tree. This helps control model complexity.
subsample	0.8	Specifies the fraction of training data to be randomly resampled during each boosting iteration. Helps prevent overfitting.
colsample_bytree	0.8	Fraction of features(columns) to be randomly sampled for each tree. Helps prevent overfitting.
gamma	1	Minimum loss reduction required to make a further partition on a leaf node. It controls the complexity of the tree by penalizing the creation of additional splits.
lambda	1	L2 regularization term on weights. It helps prevent overfitting by penalizing large weights.
alpha	0.5	L1 regularization term on weights. It helps prevent overfitting by penalizing large weights.

Upon running cross validation we found that the optimal number of rounds we should run is 32 rounds. In addition to the training data, the parameters chosen, and as well as the selected number of rounds, we created a XGBoost model and predicted our testing dataset.



```
> printMatrixInfoXGBoost(SensorData.xgboostModel.ROSED.confM, "XGBoost Model(Balanced data)")
Accuracy of XGBoost Model(Balanced data) :  92.1 %
Precision of XGBoost Model(Balanced data) :  87.89379 %
Recall of XGBoost Model(Balanced data) :  97.65 %
F1 score of XGBoost Model(Balanced data) :  92.5154 %
```

Figure 18: Evaluation of XGBoost Model

From figure X we can see that the XGBoost model has a 92.1% chance of correctly predicting whether or not a node is malicious. It also shows that when the model predicts that the node is malicious, it is correct 87.9% of the time. Meanwhile, a percentage of 97.65% for recall shows that the model is highly efficient in being able to capture a large proportion of the actual malicious nodes, even though a few of them might be non-malicious nodes. F1 score is a comprehensive metric that considers both false positives and false negatives. A score of 92.52% shows that it has good precision and recall.

LSTM:

The stacked LSTM model was used in our project, where the data was split into training and testing datasets, with a split ratio of 7:3. The model was run using Python and the deep learning frameworks, Keras, Tensorflow and Scikit-learn. The data was initially selected based on the columns we decided to continue with after the feature selection processes were completed, and it was also scaled using the scaling method from the Scikit-learn's preprocessing module. The LSTM model was constructed using two LSTM layers and a dense layer for the classification.

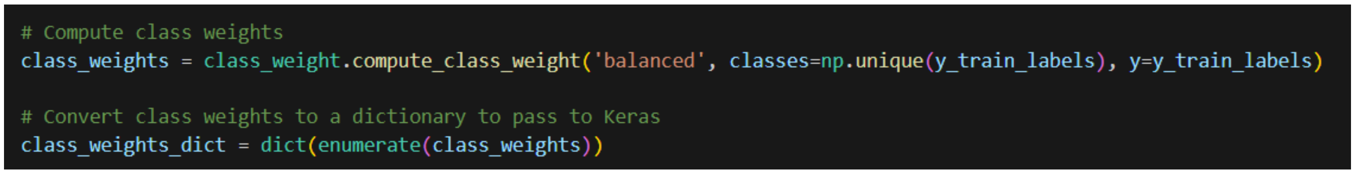
The model summary is as follows:

Model: "sequential_13"

```
_________________________________________________________________
 Layer (type)              Output Shape              Param #
=================================================================
 lstm_24 (LSTM)            (None, 1, 50)             13400
```

```
lstm_25 (LSTM)          (None, 50)          20200


dense_13 (Dense)        (None, 2)           102


=================================================================
Total params: 33702 (131.65 KB)
Trainable params: 33702 (131.65 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

An important feature used in this implementation to address the imbalance of the data is the *class_weight* from sklearn.utils and numpy. The conversion of y_train into a 1D array is needed for computing the class weights.



```
# Compute class weights
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train_labels), y=y_train_labels)

# Convert class weights to a dictionary to pass to Keras
class_weights_dict = dict(enumerate(class_weights))
```

Figure 19: Class weights computed to address the imbalance in data

This function calculates weights that can be used to balance the classes in the model training process. The purpose of this is to give more weight to the minority class, which helps in reducing the issue of imbalance in the minority class. These class weights are then converted into a dictionary format (class_weights_dict) as expected by Keras. The stacked LSTM model consists of two LSTM layers, where the **return_sequence** is first set to *True* and then *False* in the second layer. This is mainly due to the fact that since there is stacking multiple layers of LSTM, the second layer is expected to receive a sequence input in order to maintain the dependencies between the time steps of the data. After this, the second layer then mentions the **return_sequence** to *False*, as the dense layer after it expects a vector input. Hence, setting it to False condenses all the information into a single vector.

After training the model on different parameters, the model is evaluated on the test data (X_test, y_test) to calculate its accuracy and the training and validation loss per epoch is plotted.
The accuracy attained was after 20 epochs was 99.83% and the plot is as shown below:

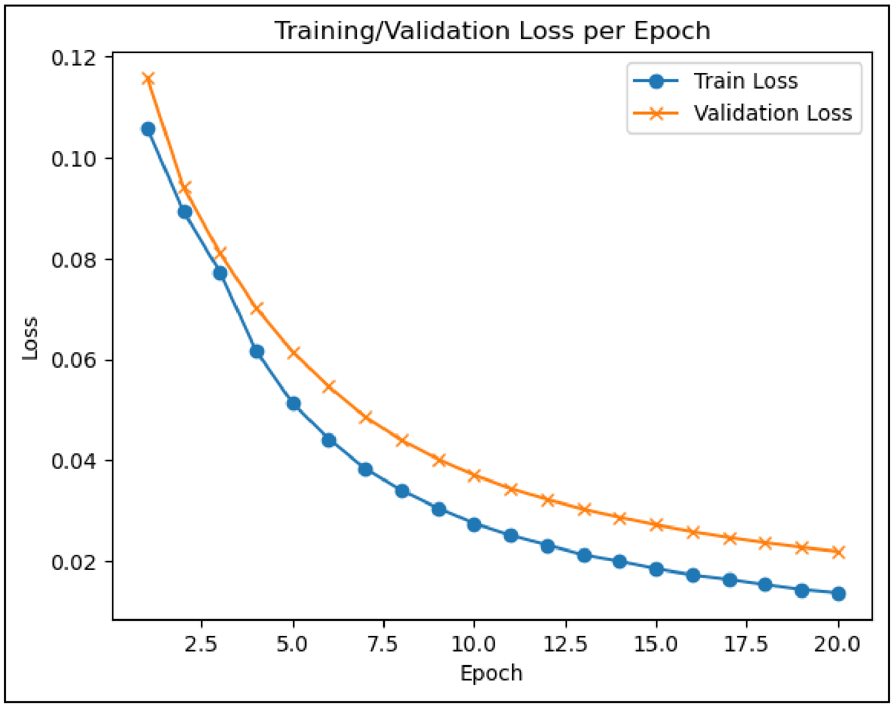


Figure 20: Training/Validation Loss per Epoch

Based on the plot, it can be seen that the model is learning well and improves its performance on both the training and validation datasets as the number of epochs increases.
Hence, to evaluate the performance of each model, the table below depicts the results gained from each of the models:

Model Used	**Accuracy**	**Precision**	**Recall**	**F1-Score**
RandomForest	91.35%	87.42%	94.22%	91.8%
SVM (Linear)	90.975%	86.503%	97.8%	91.55%
SVM (Radial)	90.55%	84.866%	98.7%	91.262%
XGBoost	92.1%	87.89%	97.65%	92.515%

Stacked LSTM	99.83%	99.91%	98.16%	99.02%

8.Conclusion

In this project, we addressed the important issue of identifying malicious sensor nodes in IoT networks using statistical models. The dataset, obtained from the IEEE DataPort website, consisted of 21 features and 10,000 samples, and we aimed to classify whether or not a certain node is malicious or not..

We employed a diverse set of machine learning models, including Random Forest, Support Vector Machines (SVM), XGBoost, and Long Short-Term Memory (LSTM) networks. Each model was trained and evaluated using appropriate performance metrics such as accuracy, precision, recall, and F1-score. Additionally, we utilized data preprocessing techniques, feature selection methods like PCA, and addressed the issue of class imbalance.

Key Findings:
XGBoost and LSTM: Both XGBoost and Stacked LSTM achieved remarkable accuracy, precision, recall, and F1-score, demonstrating their effectiveness in identifying malicious nodes.

SVM Performance: The Linear Kernel SVM and Radial Kernel SVM performed well, providing similar results and a balanced performance across key metrics.

Random Forest: While Random Forest also demonstrated good performance, it slightly lagged behind XGBoost and LSTM in terms of accuracy and F1-score.

Recommendations:
Model Selection: The choice between XGBoost, SVM (Radial/Linear Kernel), and Stacked LSTM depends on factors such as computational resources, interpretability, and the specific requirements of the application.

Validation: Models should be validated on real-world data and further fine-tuned to enhance their generalization to new scenarios.

Ensemble Methods: Considering the strengths of each model, an ensemble approach, combining the predictions of multiple models, could potentially improve overall performance.

In conclusion, with the application of advanced machine learning models, careful feature selection, and addressing class imbalance have proven effective in accurately identifying malicious sensor nodes. The high-performance metrics obtained from XGBoost and Stacked LSTM highlight their potential for enhancing network security in IoT environments.

9.References

[1] M. Singh, G. Mehta, C. Vaid, and P. Oberoi, “Detection of Malicious Node in Wireless Sensor Network Based on Data Mining,” Sep. 2012, doi: https://doi.org/10.1109/iccs.2012.24.

[2] G. Kaur and M. Singh, “Detection of black hole in Wireless Sensor Network based on Data Mining,” Sep. 2014, doi: https://doi.org/10.1109/confluence.2014.6949343.

[3] A. P. Muniyandi, R. Rajeswari, and R. Rajaram, “Network Anomaly Detection by Cascading K-Means Clustering and C4.5 Decision Tree algorithm,” Procedia Engineering, vol. 30, pp. 174–182, 2012, doi: https://doi.org/10.1016/j.proeng.2012.01.849.

[4] X. Tan et al., “Wireless Sensor Networks Intrusion Detection Based on SMOTE and the Random Forest Algorithm,” Sensors, vol. 19, no. 1, p. 203, Jan. 2019, doi: https://doi.org/10.3390/s19010203.