Real-time Cyber Intrusion detection using Network Traffic and Behaviours

Submitted by-

Kamlesh Kumar

Student ID: 017047212

I. Introduction:

Real-time Internet of Things (RT-IoT2022) [2] is a proprietary dataset derived from a real-time IoT infrastructure, meticulously curated to serve as a comprehensive resource for research in network security, particularly in the context of IoT environments. This dataset integrates a diverse range of IoT devices and sophisticated network attack methodologies, offering insights into both normal and adversarial network behaviours. With its inclusion of data from various IoT devices and simulated attack scenarios, RT-IoT2022 provides a nuanced representation of real-world network traffic scenarios. This report aims to explore the strength of various machine learning models to predict attach type with an objective to apply these algorithms in real-time for cyber-threat detection and preventions. Code Link- https://github.com/Kamlesh364/Cyber-Attack-Detection.git

II. Data Set Description:

RT-IoT2022 encompasses data collected from IoT devices such as ThingSpeak-LED, Wipro-Bulb, and MQTT-Temp, along with simulated attack scenarios involving Brute-Force SSH attacks, DDoS attacks using Hping and Slowloris, and Nmap patterns. The dataset captures bidirectional attributes of network traffic, utilizing the Zeek network monitoring tool and the Flowmeter plugin for meticulous data collection.

a) Dataset Characteristics:

- Tabular: The dataset is structured in a tabular format, enabling easy manipulation and analysis of data.
- Sequential: It captures the sequential nature of network traffic, providing insights into temporal patterns and dependencies.
- Multivariate: RT-IoT2022 includes multiple variables, representing various aspects of IoT device behaviour and network activity.

b) Subject Area:

- The dataset falls within the domain of Engineering, specifically focusing on network security and IoT infrastructure.

c) Associated Tasks:

- Researchers can utilize RT-IoT2022 for a variety of tasks including classification, regression, and clustering. These tasks enable the development and evaluation of algorithms and models for Intrusion Detection Systems (IDS) and other security solutions tailored for real-time IoT networks.

d) Feature Types:

- The features in the dataset encompass both real and categorical variables, providing a comprehensive representation of IoT device attributes and network behaviours.

e) Dataset Statistics:

- Number of Instances: 123,117 instances of network data are included in the dataset.

- Number of Features: RT-IoT2022 comprises 83 features, each offering insights into different aspects of IoT device behaviour and network traffic.

f) Potential Applications:

The RT-IoT2022 dataset serves as a valuable resource for researchers and practitioners in the field of network security. It can be utilized for:

- Developing and evaluating Intrusion Detection Systems (IDS) tailored for IoT environments.

- Investigating patterns of normal and adversarial behaviour in IoT networks.

- Benchmarking the performance of security algorithms and techniques in detecting and mitigating various types of attacks.

- Informing the design and implementation of robust and adaptive security solutions for real-time IoT infrastructures.

Dataset link: https://archive.ics.uci.edu/dataset/942/rt-iot2022

III. Dataset Visualization:

The RT-IoT2022 dataset comprises 123,117 instances and 84 features. Upon cleaning, it was observed that there were 5,195 duplicate rows, resulting in a final dataset of 117,922 instances and 84 features.

Class name	**# of instances**
DOS_SYN_Hping	90,089
Thing_Speak	7,654
ARP_poisioning	7,625
MQTT_Publish	4,142
NMAP_UDP_SCAN	2,584
NMAP_XMAS_TREE_SCAN	2,010
NMAP_OS_DETECTION	2,000
NMAP_TCP_scan	1,002
DDOS_Slowloris	533
Wipro_bulb	219
Metasploit_Brute_Force_SSH	36
NMAP_FIN_SCAN	28

Table 1: Class distribution (number of instances) for each attack type.

1. Class Label Distribution:

   The dataset contains 12 different attack types, with their respective counts as follows:

2. Feature Importance Results:

   A Random Forest Classifier was trained to determine feature importance. The results are shown in Figure 2, accordingly.

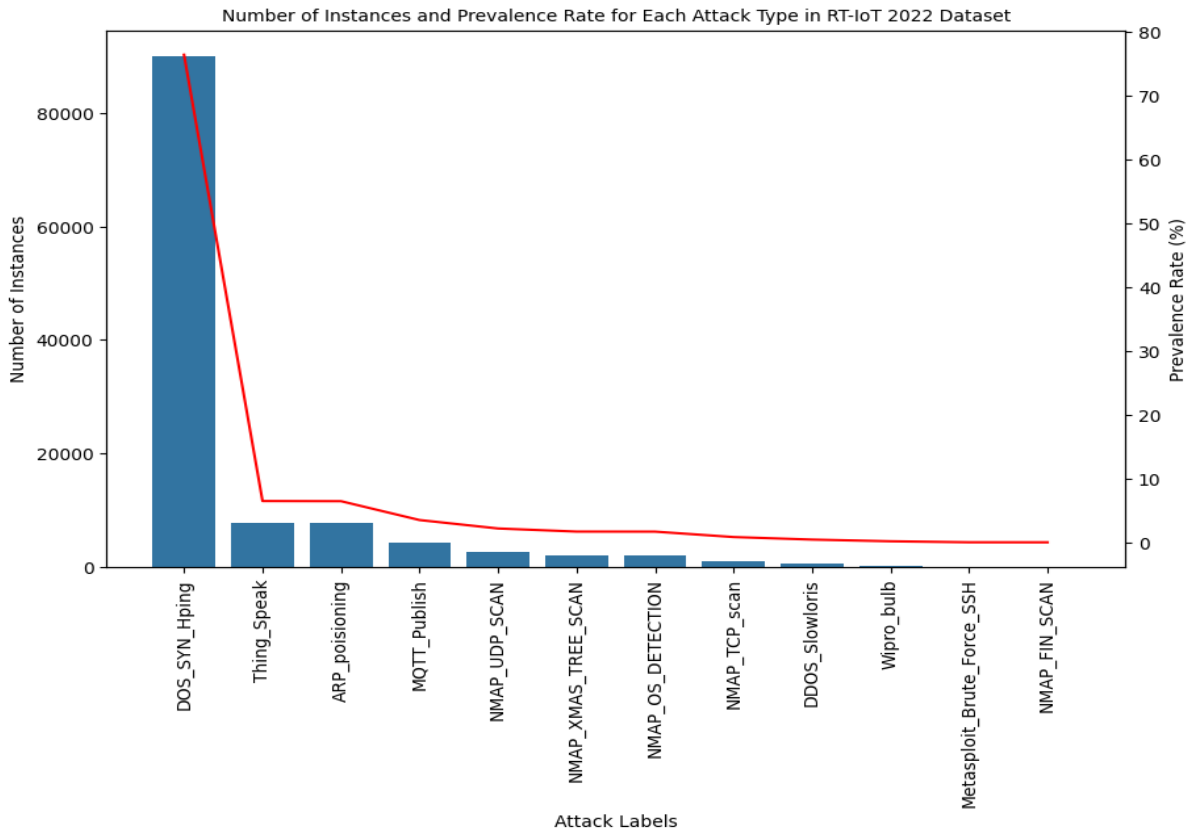

Figure 1: Class distribution and prevalence rate by visualizing number of instances and prevalence rate for each attach type.

IV. Data Set Cleaning:

This section includes various steps of data cleaning addressed below-

1. Duplicate Removal: The initial dataset contained 5,195 duplicate rows, which were identified and removed to ensure data integrity and prevent bias in subsequent analysis. The removal process resulted in a final dataset comprising 117,922 instances and 84 features.
2. Handling Missing Values: Upon examination, no missing values were found in the dataset. Therefore, there was no need for imputation or removal of rows with missing data.
3. Data Type Conversion: Before proceeding with analysis, appropriate data type conversions were applied to ensure consistency and accuracy in computations. Specifically, categorical variables were appropriately encoded, and numerical variables were ensured to be in the correct numerical format.

4. Outlier Detection and Treatment: Outliers, if present, could significantly impact model performance and analysis outcomes. Robust techniques such as IQR (Interquartile Range) method or z-score analysis were employed to detect and potentially treat outliers in numerical features. However, no specific outlier treatment was performed if outliers were within reasonable bounds and deemed to be genuine data points.

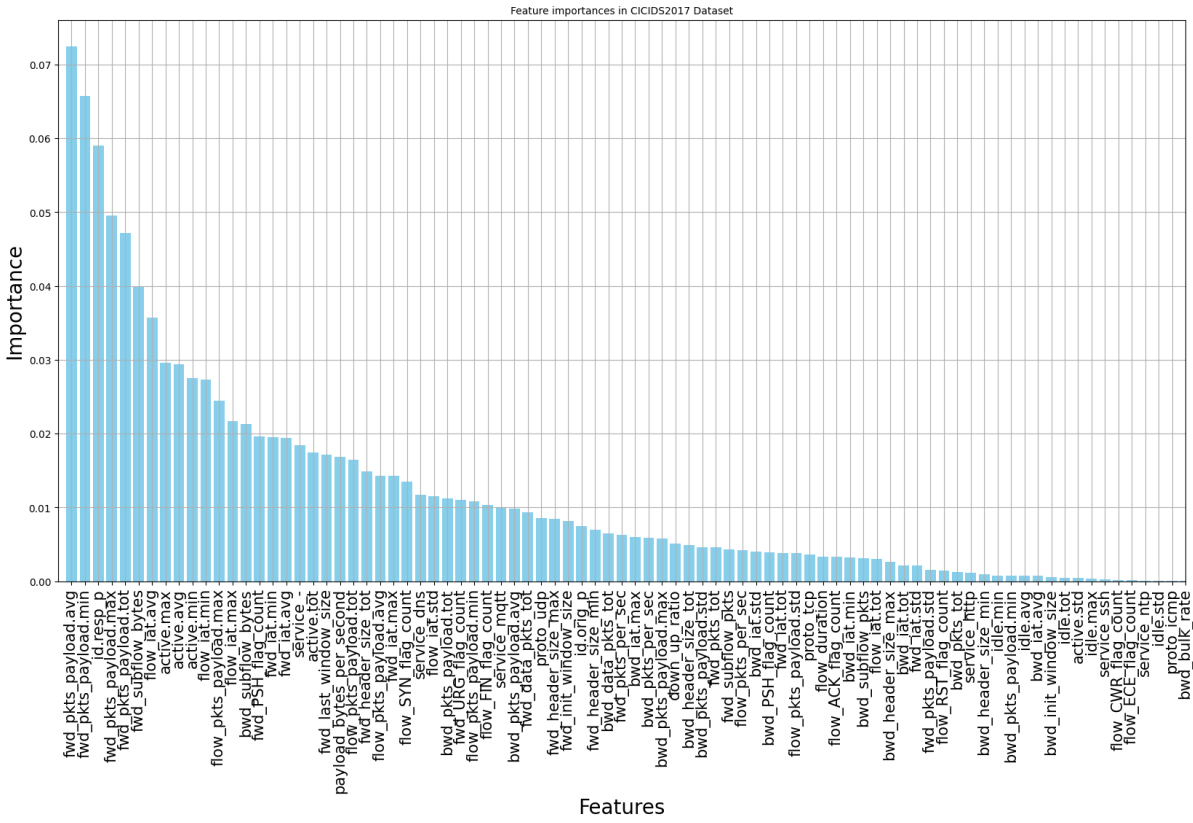

Figure 2: Feature Importance based on Random Forest Classifier training results.

5. Feature Scaling: Feature scaling is crucial for many machines learning algorithms, ensuring that all features contribute equally to the model training process. Common scaling technique Min-Max Scaling was applied to scale numerical features to a comparable range.

V. Related Work:

Recent research has emphasized unsupervised learning approaches for detecting network anomalies in edge devices, crucial for identifying attacks in resource-constrained environments. This study introduces the quantized autoencoder (QAE) model for intrusion detection systems, leveraging optimization techniques such as pruning, clustering, and integer quantization. Two variants of QAE, quantized autoencoder uint8 (QAE-u8) and quantized autoencoder float16 (QAE-f16), are proposed to address the computational constraints of edge devices. The study begins by constructing the Real-Time Internet of Things 2022 dataset, comprising normal and attack traffic, which is used to train the autoencoder model on normal

traffic. Subsequently, the model reconstructs anomaly traffic, with high reconstruction error (RE) indicating potential attacks. Performance evaluation involves comparing the accuracy, precision, recall, and F1 score of the autoencoders, QAE-u8, and QAE-f16 through extensive experimentation. Results demonstrate that QAE-u8 surpasses other models, exhibiting significant reductions in memory utilization, memory size compression, and peak CPU utilization. Consequently, the QAE-u8 model emerges as the preferred choice for deployment on resource-constrained IoT edge devices [1].

IV. Feature Extraction and Fine-tuning Feature Sets:

Several well-known feature engineering techniques were used to modify existing features and create new features. A brief description of all the techniques utilized can be found below-

1. Feature Scaling:

   - Description: Feature scaling is a technique used to standardize the range of independent variables or features in the dataset. It ensures that all features contribute equally to the analysis by bringing them to a similar scale.

   - Min-Max Scaling: Rescales features to a fixed range (typically 0 to 1) by subtracting the minimum value and dividing by the range.

   - Standardization: Centres the feature distribution around zero with a standard deviation of one by subtracting the mean and dividing by the standard deviation.

2. Feature Transformation:

   - Description: Feature transformation involves transforming the distribution of numerical features to make them more suitable for modelling. It helps to address issues like skewness and nonlinearity in the data distribution.

   - Log Transformation: Applies the natural logarithm function to numerical features to reduce skewness and make the distribution more symmetric.

3. Dimensionality Reduction:

   - Description: Dimensionality reduction techniques aim to reduce the number of features in the dataset while preserving most of the information. This helps to mitigate the curse of dimensionality, improve computational efficiency, and reduce overfitting.

   - Principal Component Analysis (PCA): Finds linear combinations of features (principal components) that capture the maximum variance in the data. It projects the data onto a lower-dimensional subspace while retaining most of the variability.

4. Feature Interaction:

- Description: Feature interaction involves creating new features by combining or interacting existing features. It allows the model to capture nonlinear relationships and interactions between different features.

- Example: Creating interaction terms by multiplying or adding two or more numerical features together.

Method	**Dataset shape**
Min-Max Scaling	(117922, 81)
Standardization	(117922, 81)
Log Transformation	(117922, 84)
PCA	(117922, 10)
Interaction Features	(117922, 85)
Domain-specific Features	(117922, 85)
Encoded Categorical Variables	(117922, 95)
Selected Features (RFE)	(117922, 10)
Target Encoded Categorical Variables	(117922, 86)

Table 2: List of modified datasets using feature engineering methods and final shapes.

5. Domain-specific Feature Engineering:

   - Description: Domain-specific feature engineering involves creating new features based on domain knowledge or insights. These features are tailored to the specific problem domain and can improve model performance by capturing relevant information.

   - Example: Creating ratios, aggregations, or temporal features based on the characteristics of the data.

6. Encoding Categorical Variables:

   - Description: Categorical variables need to be encoded into numerical format for machine learning algorithms to process them. Encoding techniques convert categorical variables into a numerical representation that can be used as input features.

   - One-Hot Encoding: Creates binary dummy variables for each category in a categorical feature, representing the presence or absence of each category.

7. Feature Selection:

   - Description: Feature selection aims to identify the most relevant features in the dataset while discarding irrelevant or redundant ones. It helps to improve model interpretability, reduce overfitting, and enhance computational efficiency.

- Recursive Feature Elimination (RFE): Recursively removes features and selects the subset that yields the best performance with a given estimator.

8. Target Encoding:

   - Description: Target encoding encodes categorical variables based on the target variable's mean value within each category. It captures target-related information in categorical features and is useful for classification tasks.

   - Example: Encoding each category in a categorical feature with the mean target value for that category.

   By utilizing the techniques mentioned above we created a set of datasets (listed in Table 2) to train multiple Machine Learning models and test their performance by calculating performance metrics like- Precision, Recall, F-1 Score (all three, class-wise and average), and Overall Classification Accuracy (Accuracy).

VIII. Model Development:

This project involves two types of model development for experimentation and analysis on RT-IoT2022 dataset. Both types include model implementations from Scikit-learn [3] python library, with and without tuning default hyperparameters provided by the framework itself. Further discussion as follows:

1. **Default Parameters:** In this subsection, we explore the performance of various machine learning models using their default parameters without any tuning.

   - Logistic Regression: Logistic Regression is a linear classification algorithm widely used for binary classification tasks. We utilized the default settings provided by scikit-learn's `LogisticRegression` class.

   - LDA/QDA (Linear Discriminant Analysis / Quadratic Discriminant Analysis): LDA and QDA are classification techniques based on Bayes' theorem. We used scikit-learn's `LinearDiscriminantAnalysis` and `QuadraticDiscriminantAnalysis` classes with their default settings.

   - KNN (K-Nearest Neighbors): KNN is a non-parametric classification algorithm that classifies a data point based on the majority class among its k nearest neighbors. We employed scikit-learn's `KNeighborsClassifier` with default settings.

   - Support Vector Machines (SVM): SVM is a powerful supervised learning algorithm used for classification tasks. We utilized scikit-learn's `SVC` class with default parameters.

- Decision Trees: Decision Trees are versatile algorithms capable of performing classification and regression tasks. We used scikit-learn's `DecisionTreeClassifier` with default settings.
- Random Forest: Random Forest is an ensemble learning method that constructs multiple decision trees and combines their predictions. We employed scikit-learn's `RandomForestClassifier` with default parameters.

2. **Fine-tunning models:**

In this subsection, we explore the performance of the machine learning models after tuning their hyperparameters using techniques such as Grid Search or Random Search.

1. Logistic Regression:
   - Penalty (L1 or L2 regularization)
   - Regularization strength (C) - [0.1, 1, 10],
2. LDA/QDA (Linear Discriminant Analysis / Quadratic Discriminant Analysis):
   - No hyperparameters to tune for LDA.
   - For QDA, regularization parameter (reg_param) is tuned.
3. KNN (K-Nearest Neighbors):
   - Number of neighbors (n_neighbors) - [3, 5, 7],
   - Distance metric (e.g., Euclidean, Manhattan, Minkowski)
4. SVM (Support Vector Machines):
   - Kernel type (linear, polynomial, radial basis function (RBF), etc.)
   - Regularization parameter (C) - [0.1, 1, 10].
   - Kernel coefficient (gamma for RBF kernel) - ["scale", "auto"].
5. Decision Trees:
   - Maximum depth of the tree (max_depth) - [None, 5, 10].
   - Minimum number of samples required to split internal node - [2, 5, 10].
   - Minimum number of samples required to be at a leaf node (min_samples_leaf) - [1, 2, 4].
   - Criterion for split (e.g., Gini impurity, entropy)
6. Random Forest:
   - Number of trees in the forest (n_estimators) - [50, 100, 200].
   - Maximum depth of the trees (max_depth) - [None, 5, 10].
   - Minimum number of samples required to split an internal node - [2, 5, 10].
   - Minimum number of samples required to be at a leaf node - [1, 2, 4].
   - Criterion for split (e.g., Gini impurity, entropy)
   - Maximum number of features to consider for split (max_features)

These are the hyperparameters we tuned for each model and each of them was trained on all datasets, original as well as, created using various feature engineering methods for a comprehensive performance analysis.

IX. Performance Metrics:

A brief description of the performance metrics we used in classification task is as following:

1. Accuracy: Accuracy measures the overall correctness of the classifier, calculated as the ratio of correctly predicted instances to the total instances.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Where:

- TP (True Positive): Number of correctly predicted positive instances.
- TN (True Negative): Number of correctly predicted negative instances.
- FP (False Positive): Number of incorrectly predicted positive instances (Type I error).
- FN (False Negative): Number of incorrectly predicted negative instances (Type II error).

2. Precision: Precision measures the ratio of correctly predicted positive observations to the total predicted positives.

$$\Pr e\,cision = \frac{TP}{TP+FP}$$

Precision is sensitive to the number of false positives.

3. Recall (Sensitivity): Recall measures the ratio of correctly predicted positive observations to all actual positives in the dataset.

$$Recall = \frac{TP}{TP+FN}$$

Recall is sensitive to the number of false negatives.

4. F1-score: F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall.

$$F1\text{-}score = \frac{2\times precision\times recall}{precision+recall}$$

F1-score reaches its best value at 1 and worst at 0. It is a useful metric when dealing with imbalanced classes.

These metrics are essential for evaluating the performance of classification models and provide insights into their effectiveness in making correct predictions.

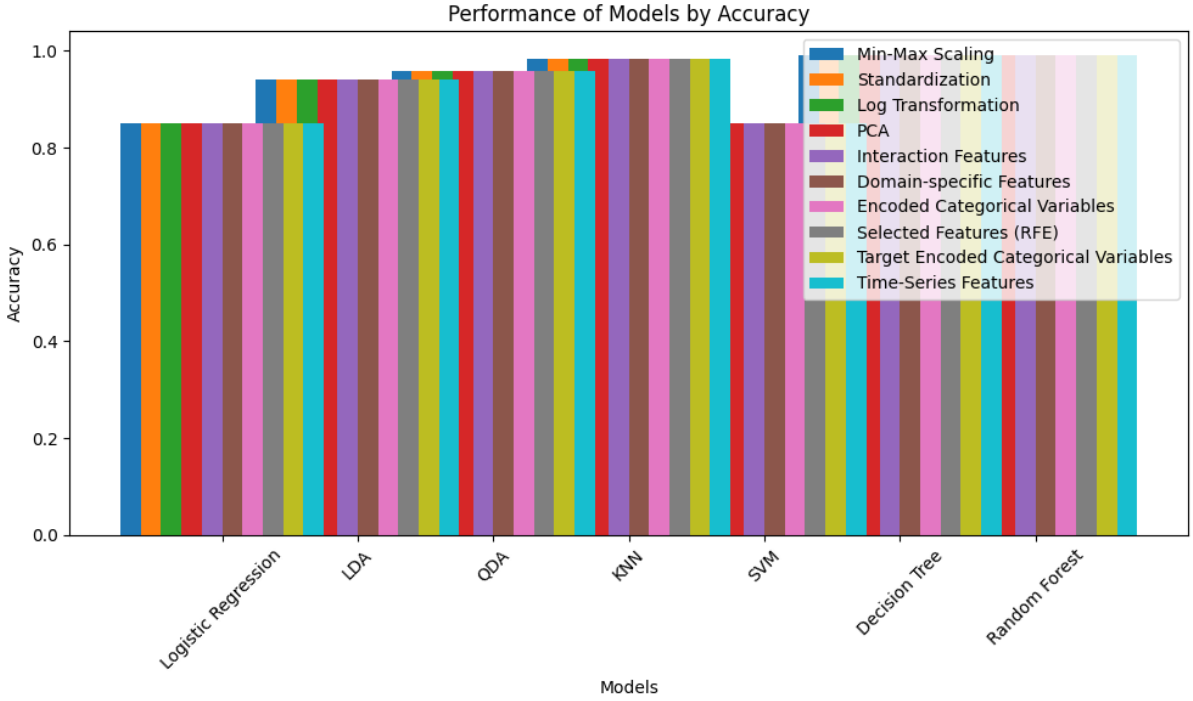

Figure 3: Performance in terms of Accuracy of all models discussed in Section VIII (with default hyperparams) on all datasets prepared in Table 2.

X. Results and Discussions:

To evaluate and compare the performance of all feature engineering methods as well as the hyperparameter tuning methodologies, we trained all datasets (shown in Table 2) on all models mentioned in section VIII (with and without hyper-params). Figure 3 shows the performance of each model, in default settings, in terms of overall accuracy on all datasets (listed in table 2) with (train:test~8:2) ratio. This figure shows that there is no significant difference between accuracies for any specific model while changing dataset, however, Random Forest outperforms all other models with Min-Max Scaling by scoring 99.07% accuracy. Moreover, if we compare the models based on precision, recall, and F1-score, Random Forest scores best precision, recall and F1-score are highest for Decision Tree, as shown in Figure 4. There is an interest conclusion to be drawn that all best metrics are obtained on Min-Max Scaling method only, which might imply that other methods are not well suitable for the RT-IoT2022 dataset and distributions.

Furthermore, after tuning some hyperparameters and including cross validation (5-Fold), we observe the performance shown in Figure 5. This experiment again supports the supremacy of Random Forest model with little change in performance metrics. We observe that Random Forest model outperforms other model with hyper-parameter tuning and scores 99.07 % accuracy and 99.06 % average 5-fold cross validation accuracy. Overall accuracy seems to be

equal to previous case, but average cross validation accuracy close to overall accuracy supports the robustness of Random Forest against different input settings. Moreover, Figure 6 shows that with mentioned hyperparameter tuning there is a possibility to get better with Precision and Recall metrics based on final applications. Figure 5 suggests that best precision decreases to 95.05%, recall and f1-scores are 92.78% and 93.30%, respectively, which are better than default settings. This is also interesting to note that best precision is still given by Random Forest and other two (recall and f1-scores) from Decision Tree for Min-Max Scaling.

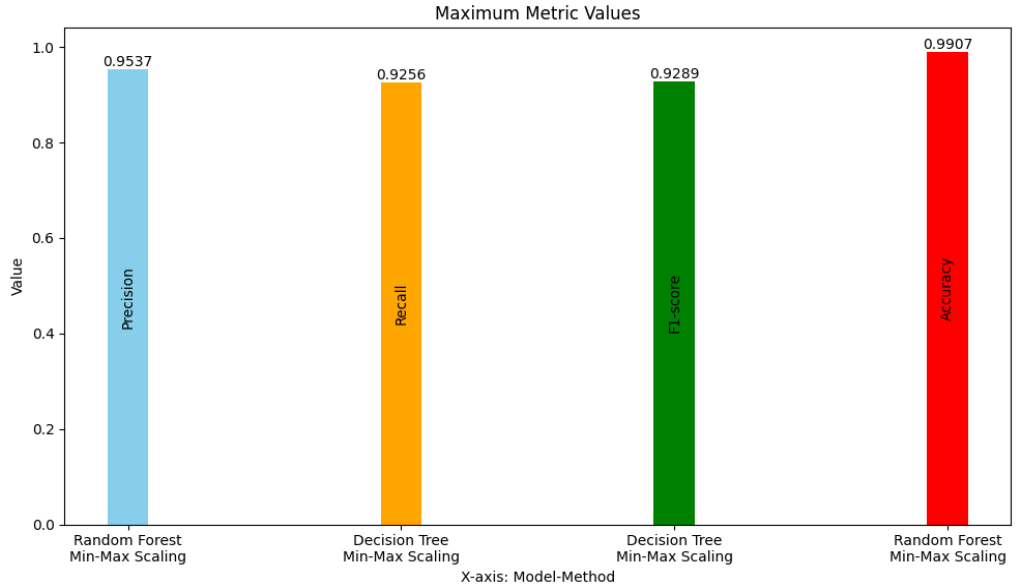

Figure 4: Best models and their performances terms of Precision, Recall, F1-score, and Accuracy amongst all models trained over all available dataset settings (with default hyperparams).

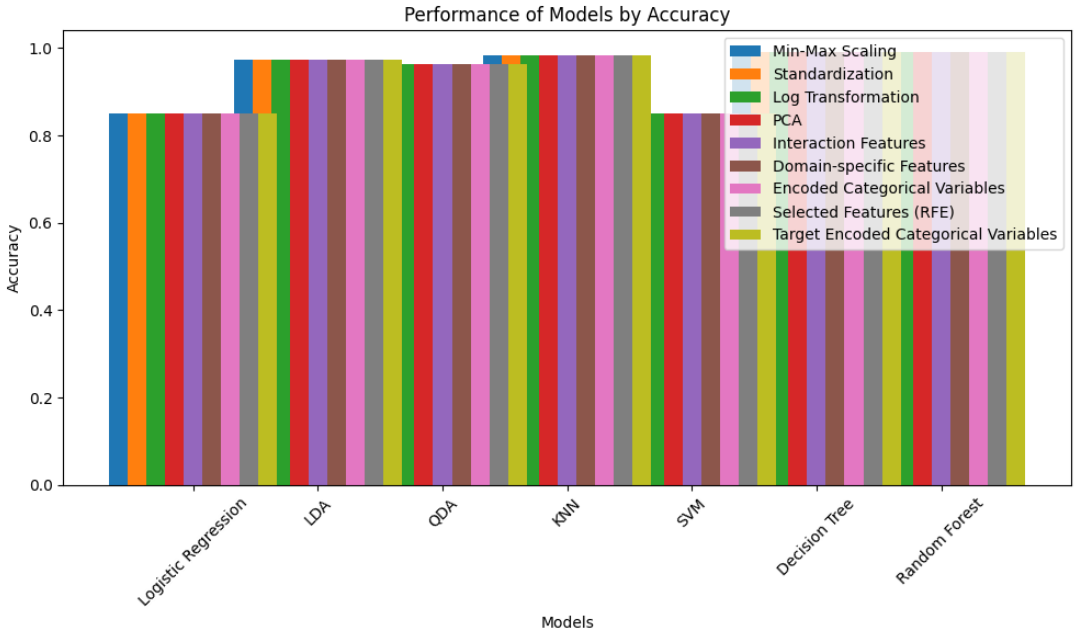

Figure 5: Performance in terms of Accuracy of all models discussed in Section VIII (with tuned hyperparams) on all datasets prepared in Table 2.

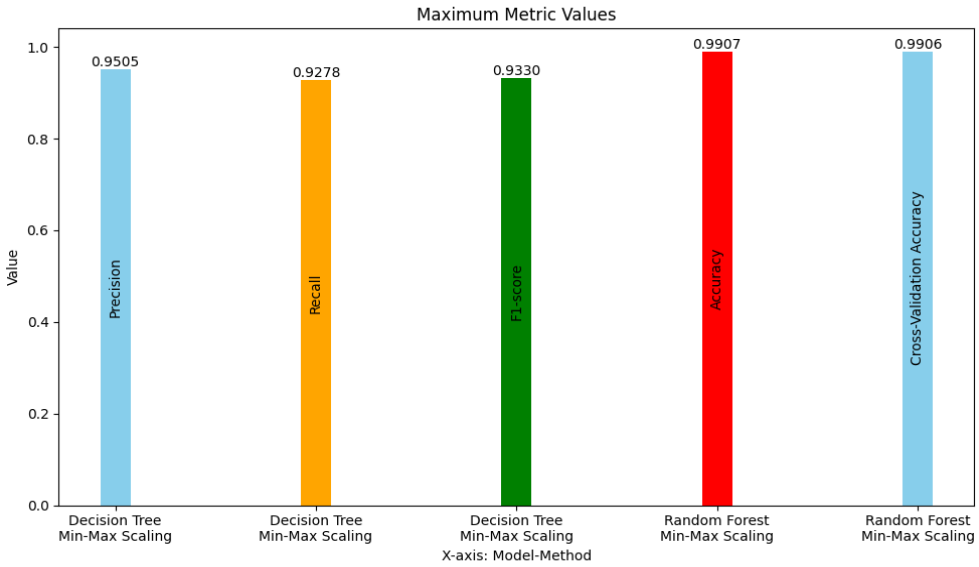

Figure 6: Best models and their performances terms of Precision, Recall, F1-score, and Accuracy amongst all models trained over all available dataset settings (with tuned hyperparams).

Figure 7 shows the confusion matrix in form of a heatmap for Random Forest Model which outperformed all models in Overall Accuracy with Min-Max Scaled dataset. The confusion matrix shows a significantly accurate classification rate amongst label 1 to 9 and 11, whereas classes 0 and 10 are more prone to be classified incorrectly. This matrix also implies that class 0 ("DOS_SYN_Hping") is close to class 8 ("DDOS_Slowloris") and 10 ("Metasploit_Brute_Force_SSH") and probable to be misclassified amongst one of them.

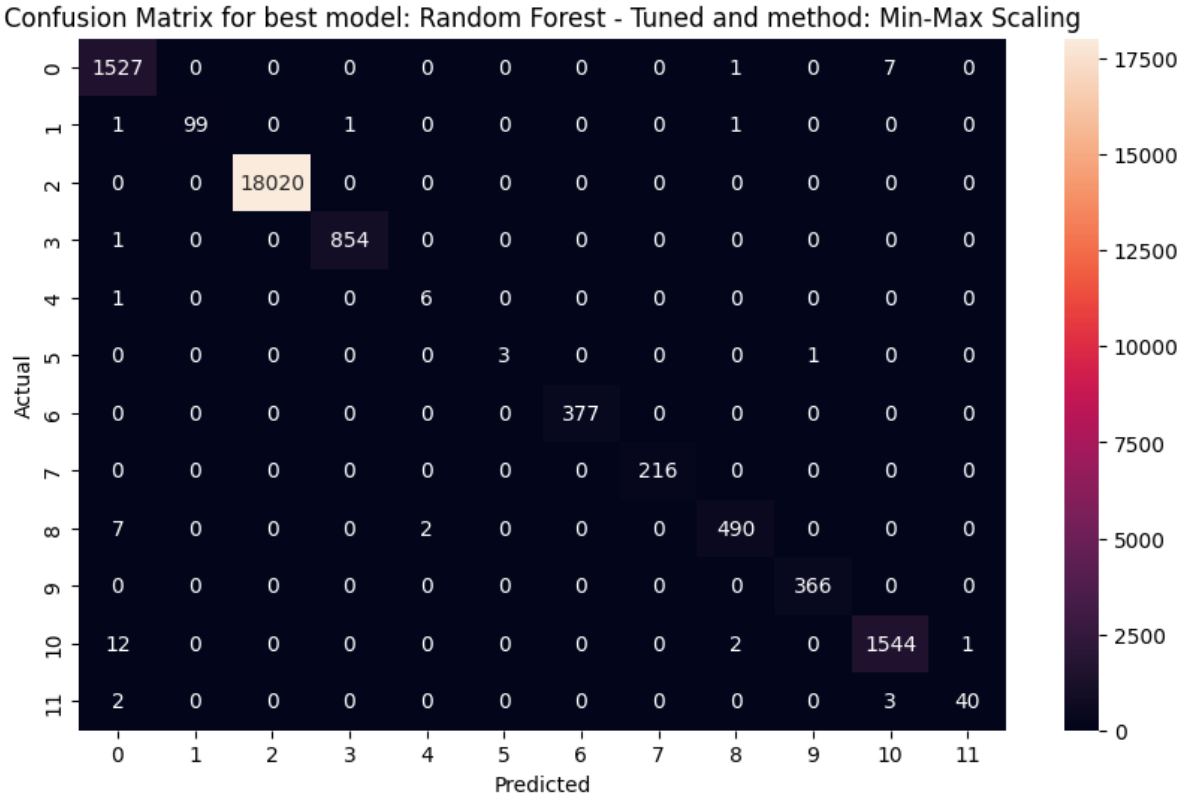

Figure 7: Best model performance on test data (20% portion of RT-IoT2022 dataset) with heatmap representation.

XI. Conclusions:

The analysis underscores the effectiveness of Random Forest and Decision Tree models, especially when applied with Min-Max Scaling, for the RT-IoT2022 dataset. These findings have significant implications for real-world applications, suggesting the potential of these models in IoT security systems for threat detection and mitigation. Future research could explore the integration of these models into practical IoT environments and assess their performance in real-time scenarios to enhance cybersecurity measures effectively. Additionally, investigating ensemble methods and deep learning architectures could further optimize model performance and robustness for comprehensive IoT security solutions.

References:

[1]. Sharmila, B.S., Nagapadma, R. Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using RT-IoT2022 dataset. *Cybersecurity* **6**, 41 (2023). doi:10.1186/s42400-023-00178-5.

[2]. S., B. and Nagapadma, Rohini. (2024). RT-IoT2022. UCI Machine Learning Repository. doi:10.24432/C5P338.

[3]. Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.